REST API□南

← □回主文档

本指南提供了OmniSS7 **REST API** 和 **Swagger UI** 的全面文档。

目录

概述

OmniSS7提供了一个REST API，用于以编程方式访问MAP（移动应用部分）操作。该API允许您：

- 发送MAP请求（SRI、SRI-for-SM、UpdateLocation等）
- 检索MAP响应
- 通过Prometheus监控系统指标

API架构

HTTP服务器配置

服务器详情

参数	值	可配置
协议	HTTP	否
IP地址	0.0.0.0（所有接口）	仅通过代码
端口	8080	仅通过代码
传输	Plug.Cowboy	否

访问**URL**: `http://[server-ip]:8080`

启用/禁用HTTP服务器

控制HTTP服务器是否启动:

```
config :omniss7,
  start_http_server: true  # 置为false以禁用
```

默认: `true`（启用）

禁用时: HTTP服务器将不会启动，REST API/Swagger UI将不可用。

Swagger UI

该API包括一个**Swagger UI**，用于交互式API文档和测试。

访问Swagger UI

URL: `http://[server-ip]:8080/swagger`

功能：

- 交互式API文档
- 测试端点的尝试功能
- 请求/响应架构
- 示例有效负载

Swagger JSON

OpenAPI规范可在以下位置获取：

URL: `http://[server-ip]:8080/swagger.json`

用例：

- 导入到Postman或其他API客户端
- 生成客户端库
- API文档自动化

API端点

所有MAP操作端点遵循模式：`POST /api/{operation}`

□□□□

□□	□□	□□	□□
`/api/sri`	POST	□□□□□□	10s
`/api/sri-for-sm`	POST	□��SM□□□□□	10s
`/api/send-auth-info`	POST	□□□□□□	10s
`/api/MT-forwardSM`	POST	□□□□□□SM	10s
`/api/forwardSM`	POST	□□SM	10s
`/api/updateLocation`	POST	□□□□	10s
`/api/prn`	POST	□□□□□□	10s
`/metrics`	GET	Prometheus□□	N/A
`/swagger`	GET	Swagger UI	N/A
`/swagger.json`	GET	OpenAPI□□	N/A

□□: □□MAP□□□□□□□□□□□□**10**□□□□

SendRoutingInfo (SRI)

□□□□□□□□□□□□□□□□□□□□□□□□□□

□□: `POST /api/sri`

□□□:

```
{
  "msisdn": "1234567890",
  "gmsc": "5551234567"
}
```

参数:

字段	类型	必需	描述
`msisdn`	字符串	是	被叫方MSISDN
`gmsc`	字符串	是	网关MSC全球标题

响应 (200 OK):

```
{
  "result": {
    "imsi": "001001234567890",
    "msrn": "5551234999",
    "vlr_number": "5551234800",
    ...
  }
}
```

错误 (504 Gateway Timeout):

```
{
  "error": "timeout"
}
```

cURL示例:

```
curl -X POST http://localhost:8080/api/sri \
  -H "Content-Type: application/json" \
  -d '{
    "msisdn": "1234567890",
    "gmsc": "5551234567"
  }'
```

SendRoutingInfoForSM (SRI-for-SM)

检索用于将SMS发送到移动用户的路由信息。

端点: `POST /api/sri-for-sm`

请求体:

```
{
  "msisdn": "1234567890",
  "service_center": "5551234567"
}
```

参数:

字段	类型	必需	描述
`msisdn`	字符串	是	目的地MSISDN
`service_center`	字符串	是	服务中心全球标题

响应 (200 OK):

```
{
  "result": {
    "imsi": "001001234567890",
    "msc_number": "5551234800",
    "location_info": {...},
    ...
  }
}
```

cURL示例:

```
curl -X POST http://localhost:8080/api/sri-for-sm \
  -H "Content-Type: application/json" \
  -d '{
    "msisdn": "1234567890",
    "service_center": "5551234567"
  }'
```

SendAuthenticationInfo

请求订阅者的认证向量。

端点: `POST /api/send-auth-info`

请求体:

```
{
  "imsi": "001001234567890",
  "vectors": 3
}
```

参数:

字段	类型	必需	描述
`imsi`	字符串	是	订阅者IMSI
`vectors`	整数	是	要生成的认证向量数量

响应 (200 OK):

```
{
  "result": {
    "authentication_sets": [
      {
        "rand": "0123456789ABCDEF...",
        "xres": "...",
        "ck": "...",
        "ik": "...",
        "autn": "..."
      }
    ],
    ...
  }
}
```

cURL示例:

```
curl -X POST http://localhost:8080/api/send-auth-info \
  -H "Content-Type: application/json" \
  -d '{
    "imsi": "001001234567890",
    "vectors": 3
  }'
```

MT-ForwardSM

将移动终止SMS发送给订阅者。

端点: `POST /api/MT-forwardSM`

请求体:

```
{
  "imsi": "001001234567890",
  "destination_service_centre": "5551234567",
  "originating_service_center": "5551234568",
  "smsPDU": "0001000A8121436587F900001C48656C6C6F20576F726C64"
}
```

参数:

字段	类型	必需	描述
`imsi`	字符串	是	目的地订阅者IMSI
`destination_service_centre`	字符串	是	目的地服务中心GT
`originating_service_center`	字符串	是	发起服务中心GT
`smsPDU`	字符串	是	以十六进制格式表示的SMS TPDU

注意: `smsPDU`必须是十六进制编码的字符串（大写或小写）。

响应 (200 OK):

```
{
  "result": {
    "delivery_status": "success",
    ...
  }
}
```

cURL示例:

```
curl -X POST http://localhost:8080/api/MT-forwardSM \
  -H "Content-Type: application/json" \
  -d '{
    "imsi": "001001234567890",
    "destination_service_centre": "5551234567",
    "originating_service_center": "5551234568",
    "smsPDU": "0001000A8121436587F900001C48656C6C6F20576F726C64"
  }'
```

ForwardSM

转发SMS消息（来自订阅者的MO-SMS）。

端点：`POST /api/forwardSM`

请求体：与MT-ForwardSM相同

cURL示例：

```
curl -X POST http://localhost:8080/api/forwardSM \
  -H "Content-Type: application/json" \
  -d '{
    "imsi": "001001234567890",
    "destination_service_centre": "5551234567",
    "originating_service_center": "5551234568",
    "smsPDU": "0001000A8121436587F900001C48656C6C6F20576F726C64"
  }'
```

UpdateLocation

通知HLR订阅者位置更改（VLR注册）。

端点：`POST /api/updateLocation`

请求体：

```
{
  "imsi": "001001234567890",
  "vlr": "5551234800"
}
```

参数:

字段	类型	必需	描述
`imsi`	字符串	是	订阅者IMSI
`vlr`	字符串	是	VLR全球标题地址

响应 (200 OK):

```
{
  "result": {
    "hlr_number": "5551234567",
    "subscriber_data": {...},
    ...
  }
}
```

注意: 在HLR模式下，这会触发带有每个ISD 10秒超时的InsertSubscriberData (ISD)序列。

cURL示例:

```
curl -X POST http://localhost:8080/api/updateLocation \
  -H "Content-Type: application/json" \
  -d '{
    "imsi": "001001234567890",
    "vlr": "5551234800"
  }'
```

ProvideRoamingNumber (PRN)

请求MSRN（移动台漫游号码）以进行呼叫路由到漫游订阅者。

端点：`POST /api/prn`

请求体：

```
{
  "msisdn": "1234567890",
  "gmsc": "5551234567",
  "msc_number": "5551234800",
  "imsi": "001001234567890"
}
```

参数：

字段	类型	必需	描述
`msisdn`	字符串	是	订阅者MSISDN
`gmsc`	字符串	是	网关MSC GT
`msc_number`	字符串	是	订阅者的MSC号码
`imsi`	字符串	是	订阅者IMSI

响应 (200 OK):

```
{
  "result": {
    "msrn": "5551234999",
    ...
  }
}
```

cURL示例：

```
curl -X POST http://localhost:8080/api/prn \
  -H "Content-Type: application/json" \
  -d '{
    "msisdn": "1234567890",
    "gmsc": "5551234567",
    "msc_number": "5551234800",
    "imsi": "001001234567890"
  }'
```

身份验证

当前状态：API 不需要身份验证。

安全考虑：

- API旨在用于内部/受信网络
- 考虑使用防火墙规则限制访问
- 对于生产部署，考虑实施身份验证中间件

响应格式

所有响应使用**JSON**格式。

成功响应

HTTP状态：200 OK

结构：

```
{
  "result": {
    // 作特定的响应数据
  }
}
```

错误响应

HTTP状态:

- 400 Bad Request - 请求体无效JSON
- 504 Gateway Timeout - MAP请求超时（10秒）
- 404 Not Found - 无效端点

结构:

```
{
  "error": "timeout"
}
```

或

```
{
  "error": "invalid request"
}
```

错误处理

常见错误

错误	**HTTP代码**	描述	解决方案
无效JSON	400	请求体不是有效的JSON	检查JSON语法
缺少字段	400	缺少必需字段	包含所有必需参数
超时	504	MAP请求超过10秒超时	检查M3UA连接，HLR/VLR可用性
未找到	404	无效端点	检查端点URL

超时行为

所有MAP请求都有一个硬编码的**10**秒超时：

1. 请求发送到MapClient GenServer
2. 等待响应最多10秒
3. 如果没有响应 → 回504 Gateway Timeout
4. 如果收到响应 → 回200 OK和结果

故障排除超时：

- 检查M3UA连接状态（Web UI → M3UA页面）
- 验证网络元素（HLR/VLR/MSC）是否可达
- 检查路由配置
- 查看SS7事件日志以查找错误

标（**Prometheus**）

该API公开Prometheus指标以供监控。

标端点

URL: `http://[server-ip]:8080/metrics`

格式：Prometheus文本格式

示例输出：

```
# HELP map_requests_total Total MAP requests
# TYPE map_requests_total counter
map_requests_total{operation="sri"} 42
map_requests_total{operation="sri_for_sm"} 158
map_requests_total{operation="updateLocation"} 23

# HELP cap_requests_total Total CAP requests
# TYPE cap_requests_total counter
cap_requests_total{operation="initialDP"} 87
cap_requests_total{operation="requestReportBCSMEvent"} 91

# HELP map_request_duration_milliseconds Duration of MAP
request/responses in ms
# TYPE map_request_duration_milliseconds histogram
map_request_duration_milliseconds_bucket{operation="sri",le="10"}
5
map_request_duration_milliseconds_bucket{operation="sri",le="50"}
12
map_request_duration_milliseconds_bucket{operation="sri",le="100"}
35
...

# HELP map_pending_requests Number of pending MAP TID waiters
# TYPE map_pending_requests gauge
map_pending_requests 3
```

可用□标

□标	类型	标签	描述
`map_requests_total`	计数器	`operation`	按操作类型的MAP请求总数
`cap_requests_total`	计数器	`operation`	按操作类型的CAP请求总数
`map_request_duration_milliseconds`	直方图	`operation`	请求持续时间（毫秒）
`map_pending_requests`	指标	-	待处理的MAP事务数量

Prometheus配置

添加到您的`prometheus.yml`:

```
scrape_configs:
  - job_name: 'omniss7'
    static_configs:
      - targets: ['server-ip:8080']
    metrics_path: '/metrics'
    scrape_interval: 15s
```

示例请求

Python示例

```python
import requests
import json

# SRI-for-SM请求
url = "http://localhost:8080/api/sri-for-sm"
payload = {
    "msisdn": "1234567890",
    "service_center": "5551234567"
}

response = requests.post(url, json=payload, timeout=15)

if response.status_code == 200:
    result = response.json()
    print(f"成功: {result}")
elif response.status_code == 504:
    print("超时 - 网络没有响应")
else:
    print(f"错误: {response.status_code} - {response.text}")
```

JavaScript示例

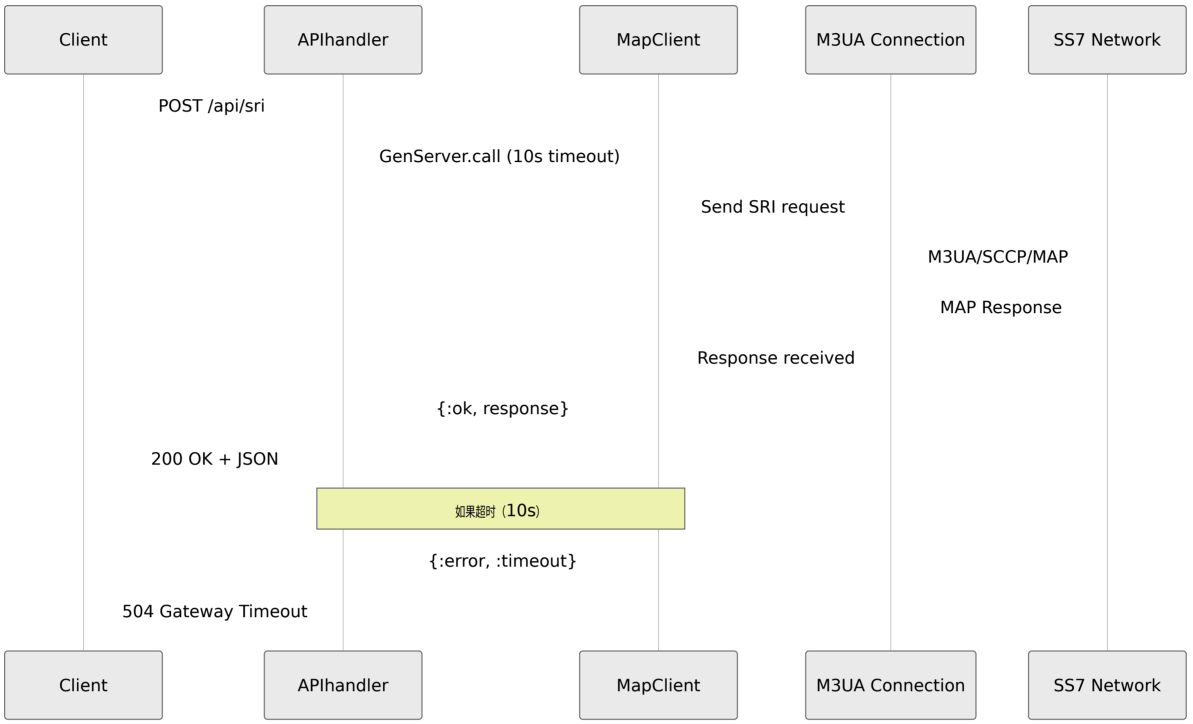
```
const axios = require('axios');

async function sendSRI() {
  try {
    const response = await
axios.post('http://localhost:8080/api/sri', {
      msisdn: '1234567890',
      gmsc: '5551234567'
    }, {
      timeout: 15000
    });

    console.log('成功:', response.data);
  } catch (error) {
    if (error.code === 'ECONNABORTED') {
      console.error('超时 - 网络没有响应');
    } else {
      console.error('错误:', error.response?.data || error.message);
    }
  }
}

sendSRI();
```

Bash/cURL示例

```
#!/bin/bash

# UpdateLocation请求
response=$(curl -s -w "\n%{http_code}" -X POST
http://localhost:8080/api/updateLocation \
  -H "Content-Type: application/json" \
  -d '{
    "imsi": "001001234567890",
    "vlr": "5551234800"
  }')

http_code=$(echo "$response" | tail -n 1)
body=$(echo "$response" | sed '$d')

if [ "$http_code" -eq 200 ]; then
  echo "成功: $body"
elif [ "$http_code" -eq 504 ]; then
  echo "超时 - 网络没有响应"
else
  echo "错误 $http_code: $body"
fi
```

流程图

API请求流程



总结

OmniSS7 REST API提供:

- **MAP**操作 - 完全支持SRI、SRI-for-SM、UpdateLocation、SMS交付、认证
- **Swagger UI** - 交互式API文档和测试
- **Prometheus**标 - 监控和可观察性
- □□□□□ - 所有MAP请求的10秒超时
- **HTTP**服务器 - 在8080端口运行（可通过`start_http_server`配置）

有关Web UI访问，请参见Web UI指南。

有关配置详细信息，请参见配置参考。

技术参考（附录）

← 回主文档

SS7协议和OmniSS7实现的技术参考。

SS7协议栈

MAP操作码

操作	操作码	目的
updateLocation	2	注册用户位置
cancelLocation	3	从VLR注销
provideRoamingNumber	4	请求MSRN
sendRoutingInfo	22	查询呼叫路由
mt-forwardSM	44	将SMS发送给用户
sendRoutingInfoForSM	45	查询SMS路由
mo-forwardSM	46	从用户转发SMS
sendAuthenticationInfo	56	请求认证向量

TCAP消息类型

- **BEGIN** - 开始事务
- **CONTINUE** - 事务中
- **END** - 最终响应
- **ABORT** - 取消事务

SCCP寻址

全球标题格式

- **E.164** - 国际电话号码（例如，447712345678）
- **E.212** - IMSI格式（例如，234509876543210）
- **E.214** - 点码格式

子系统编号（SSN）

- **SSN 6**: HLR
- **SSN 7**: VLR
- **SSN 8**: MSC/SMSC
- **SSN 9**: GMLC
- **SSN 10**: SGSN

SMS TPDU

消息类型

- **SMS-DELIVER** (MT) - 网络到移动
- **SMS-SUBMIT** (MO) - 移动到网络
- **SMS-STATUS-REPORT** - 投递状态
- **SMS-COMMAND** - 远程命令

字符编码

- **GSM7** - 7位GSM字母表（每条SMS 160个字符）
- **UCS2** - 16位Unicode（每条SMS 70个字符）
- **8-bit** - 二进制数据（每条SMS 140字节）

M3UA状态

- **DOWN** - 无SCTP连接
- **CONNECTING** - SCTP连接中
- **ASPUP_SENT** - 等待ASPUP确认
- **INACTIVE** - ASP已启动但未激活
- **ASPAC_SENT** - 等待ASPAC确认
- **ACTIVE** - 准备接收流量

常见SS7点码

点码通常为14位（ITU）或24位（ANSI）值。

示例格式（**ITU**）：

- 网络：3位
- 集群：8位
- 成员：3位

SCCP错误代码

- **0** - 地址无翻译
- **1** - 特定地址无翻译
- **2** - 子系统拥塞
- **3** - 子系统故障

- **4** - 用户未装备
- **5** - MTP故障
- **6** - 网络拥塞
- **7** - 无资格
- **8** - 消息传输错误

MAP错误代码

代码	错误	描述
1	unknownSubscriber	用户不在HLR中
27	absentSubscriber	用户不可达
34	systemFailure	网络故障
35	dataMissing	所需数据不可用
36	unexpectedDataValue	参数值无效

相关文档

- ← 回主文档
- STP指南
- MAP客户端指南
- 短信中心指南
- HLR指南
- 常见功能

CAMEL 请求构建器 - 实施摘要

概述

创建了一个新的 LiveView 组件，用于构建和发送 CAMEL/CAP 请求以进行测试。这提供了一个交互式 UI，用于创建 InitialDP 和其他 CAMEL 操作。

新组件

1. CAMEL 请求构建器 LiveView

功能：

- 基于表单的交互式 UI，用于构建 CAMEL 请求
- 支持多种请求类型：
  - **InitialDP** - 初始检测点（呼叫设置通知）
  - **Connect** - 将呼叫连接到目的地

    - **ReleaseCall** - □□/□□□□
    - **RequestReportBCSMEvent** - □□□□□□
    - **Continue** - □□□□□□
    - **ApplyCharging** - □□□□□□□/□□□□□□

□□□□□

- □□□□□□□□□□
- □□□□□□□□□□□□□□□□
- □□ SCCP/M3UA □□□□□□□□□
    - □□/□□□□□□□
    - SSN□□□□□□□□□
    - OPC/DPC□□□□□□
- □□□□□□□□□ 20 □□□□
- □□ OTID □□□□□□
- □□/□□□□□
- □□□□□□

□□□ `/camel_request`

2. □□□ EventLog □□□ CAMEL

□□□□

- `paklog_camel/2` - □□□ CAMEL/CAP □□□□□□
- `lookup_cap_opcode_name/1` - CAP □□□□□
- `find_cap_opcode/1` - □ JSON □□□ CAP □□□
- `extract_cap_tids/1` - □ CAP □□□□□ OTID/DTID
- `format_cap_to_json/1` - □ CAP PDU □□□ JSON □□

□□□ **CAP** □□□□

```
0  => "initialDP"
5  => "connect"
6  => "releaseCall"
7  => "requestReportBCSMEvent"
8  => "eventReportBCSM"
10 => "continue"
13 => "furnishChargingInformation"
35 => "applyCharging"
... (共 47 个操作)
```

功能:

- 所有 CAMEL 请求/响应的 JSON 日志记录
- 自动 TCAP 动作检测（开始/继续/结束/中止）
- SCCP 地址提取
- 对格式错误消息的错误处理
- 后台任务处理（非阻塞）
- 事件前缀为 "CAP:" 以便于过滤

3. 更新的 CapClient

更改:

- 为传入和传出消息添加了 `paklog_camel/2` 调用
- 双重日志记录: 同时记录 MAP (`paklog`) 和 CAP (`paklog_camel`) 以保持兼容性
- 传出消息记录在 `sccp_m3ua_maker/2` 中
- 传入消息记录在 `handle_payload/1` 中

配置

新的 LiveView 页面已添加到运行时配置中:

```
# 文件: config/runtime.exs

config :control_panel,
  use_additional_pages: [
    {SS7.Web.EventsLive, "/events", "SS7 事件"},
    {SS7.Web.TestClientLive, "/client", "SS7 客户端"},
    {SS7.Web.M3UAStatusLive, "/m3ua", "M3UA"},
    {SS7.Web.HlrLinksLive, "/hlr_links", "HLR 链接"},
    {SS7.Web.CAMELSessionsLive, "/camel_sessions", "CAMEL 会话"},
    {SS7.Web.CAMELRequestLive, "/camel_request", "CAMEL 请求构建器"}
  ],
  page_order: ["/events", "/client", "/m3ua", "/hlr_links",
               "/camel_sessions", "/camel_request",
               "/application", "/configuration"]
```

使用

访问请求构建器

1. 导航到: `https://your-server:8087/camel_request`
2. 从下拉菜单中选择请求类型
3. 填写所需参数
4. 可选地展开“高级 SCCP/M3UA 选项”以进行微调
5. 点击“发送 [RequestType] 请求”

请求流程

InitialDP（新呼叫）

1. 设置服务密钥（例如，100）
2. 设置主叫号码（A-Party）
3. 设置被叫号码（B-Party）
4. 发送请求 → 生成新的 OTID
5. OTID 存储在会话中以供后续请求使用

后续请求（Connect, ReleaseCall 等）

1. 必须具有来自 InitialDP 的活动 OTID
2. 请求自动使用存储的 OTID
3. 如果没有活动 OTID，则显示警告

请求参数

InitialDP:

- 服务密钥（整数）
- 主叫号码（ISDN 格式）
- 被叫号码（ISDN 格式）

Connect:

- 目的号码（呼叫路由到哪里）

ReleaseCall:

- 原因代码（16 = 正常，17 = 忙，31 = 未指定）

RequestReportBCSMEvent:

- BCSM 事件（以逗号分隔：oAnswer, oDisconnect 等）

Continue:

- 无参数（使用活动 OTID）

ApplyCharging:

- 持续时间（秒，1-864000） - 在采取行动之前的最大呼叫持续时间
- 超时释放（布尔值） - 持续时间到期时是否释放呼叫

高级选项

SCCP 地址：

- 被叫方 GT（全球标题）
- 主叫方 GT

- 被叫 SSN（默认 146 = gsmSSF）
- 主叫 SSN（默认 146）

M3UA 点码：

- OPC（发起点码，默认 5013）
- DPC（目的点码，默认 5011）

JSON 日志记录

所有 CAMEL 消息现在以 JSON 格式记录在事件日志中，包含：

- 方向：传入/传出
- **TCAP** 动作：开始/继续/结束/中止
- **CAP** 操作：例如，“CAP:initialDP”，“CAP:connect”
- **SCCP** 地址：被叫/主叫方信息
- **TIDs**：OTID/DTID 用于关联
- 完整消息：JSON 编码的 CAP PDU

示例日志条目

```
{
  "map_event": "CAP:initialDP",
  "direction": "outgoing",
  "tcap_action": "Begin",
  "otid": "A1B2C3D4",
  "sccp_called": {
    "SSN": 146,
    "GlobalTitle": {
      "Digits": "55512341234",
      "NumberingPlan": "isdn_tele",
      "NatureOfAddress_Indicator": "international"
    }
  },
  "event_message": "{ ... full CAP PDU ... }"
}
```

请求历史

UI 显示最后 20 个请求，包含：

- 时间戳
- 请求类型（带有颜色编码的徽章）
- OTID（前 8 个十六进制字符）
- 状态（已发送/错误）
- 消息大小（以字节为单位）

会话跟踪

当前会话信息面板：

- 显示活动 OTID
- 显示最后请求的字节大小
- 仅在会话处于活动状态时可见

测试工作流程

1. 开始新呼叫：
    - 发送 InitialDP → 获取 OTID
    - 系统创建会话
2. 控制呼叫：
    - 发送 RequestReportBCSMEvent → 请求通知
    - 发送 ApplyCharging → 设置呼叫持续时间限制（例如，290 秒）
    - 发送 Connect → 路由到目的地
    - 或者发送 ReleaseCall → 终止
3. 查看结果：
    - 检查请求历史
    - 监控 CAMEL 会话页面

- 审查带有 "CAP:" 前缀的事件日志

ApplyCharging - 呼叫持续时间控制

概述

ApplyCharging 操作允许您设置最大呼叫持续时间，并在该持续时间到期时可���地释放呼叫。这通常用于预付费计费场景或对呼叫施加时间限制。

用例

- 预付费计费：根据用户余额限制呼叫持续时间
- 基于时间的计费：强制执行定期计费间隔
- 资源管理：防止呼叫无限期进行
- **OCS** 集成：与在线计费系统协调以进行实时信用控制

参数

持续时间（**maxCallPeriodDuration**）

- 类型：整数（1-864000 秒）
- 描述：呼叫可以运行的最大秒数，直到计时器到期
- 示例：
  - `60` = 1 分钟
  - `290` = 4 分钟 50 秒（常见测试值）
  - `3600` = 1 小时
  - `86400` = 24 小时

超时释放（**releaseIfDurationExceeded**）

- 类型：布尔值（true/false）
- 默认：true
- 描述：持续时间到期时发生的情况：
  - `true`：自动释放/断开呼叫
  - `false`：发送通知但保持呼叫活动（允许 gsmSCF 采取行动）

消息结构

ApplyCharging 消息编码为 TCAP Continue，包含：

- **TCAP**：继续消息（使用现有事务）
- 操作码：35（applyCharging）
- 参数：ApplyChargingArg 包含：
    - `aChBillingChargingCharacteristics`：基于时间的计费信息
        - `timeDurationCharging`：最大持续时间和释放标志
    - `partyToCharge`：哪个方被计费（默认：sendingSideID）

示例用法

场景：预付费呼叫，限制为 5 分钟

1. 发送 **InitialDP** 开始呼叫监控

    ```
    服务密钥：100
    主叫：447700900123
    被叫：447700900456
    → OTID：A1B2C3D4
    ```

2. 发送 **ApplyCharging** 设置 5 分钟限制

    ```
    持续时间：300（秒）
    超时释放：true
    → 使用 OTID：A1B2C3D4
    ```

3. 发送 **Connect** 完成呼叫

    ```
    目的地：447700900456
    → 使用 OTID：A1B2C3D4
    ```

4. 5 分钟后（300 秒）：
    - 呼叫由网络自动释放

- gsmSCF □□□□□□

□□□□

1. □□□ **Connect** □□□□ **ApplyCharging**
   - □□□□□□□□□□□□□□□□□□□□
   - □□□□□□□□□□
2. □ **RequestReportBCSMEvent** □□□□
   - □□ `oAnswer` □ `oDisconnect` □□
   - □□□□□□□□□□□□□
   - □□□□□□□□□□□□□□□
3. □□□□□□□□□□
   - □□□□□□□□□□□□□□□□□□□
   - □□□□□□□□□□□□□□□□□□□□□□□
   - □□□□□□□ 60-300 □□□□□□□□
4. □□□□□□□□
   - □□ `release=false`□□□□□□□□□□□□□□□
   - □□□□□□□□□□□□□□□□□□□

□□□□

□□□□□

- □□□□ **OTID**□□□□□□ InitialDP
- □□□□□□□□□□□ 1-864000 □
- □□□□□□□□ SSF □□□□□□□□ ApplyCharging
- □□□□□□□□□□□□□□□□□□□□□ 1 □□□□□□□□□□□

□□

□□□□□□□□ ApplyCharging □□□

- 请求历史：显示已发送的 ApplyCharging 请求
- 事件日志：搜索 "CAP:applyCharging"
- **CAMEL** 会话：监控应用了计费的活动会话
- **TCAP** 跟踪：调试编码/解码问题

实施细节

状态管理

- LiveView 分配跟踪表单状态
- OTID 存储在套接字分配中
- 请求历史限制为 20 条记录
- 自动刷新禁用（仅手动发送）

请求生成

- 使用现有的 `CapRequestGenerator` 模块
- 构建正确的 TCAP/CAP 结构
- 使用 `:TCAPMessages` 编解码器进行编码
- 通过 `CapClient.sccp_m3ua_maker/2` 包装在 SCCP 中

发送机制

- 通过 M3UA 发送到 `:camelgw_client_asp`
- 使用路由上下文 1
- 自动 SCCP/M3UA 封装

错误处理

- 表单验证与用户反馈
- 优雅处理缺失的 OTID
- UI 中显示解析错误
- 编码失败记录

未来增强

潜在的补充：

1. 请求模板/预设
2. 响应关联和显示
3. 呼叫流程可视化
4. 会话详细信息深入
5. 导出请求历史
6. 负载测试（批量请求）
7. 生成消息的 PCAP 导出
8. CAP 参数验证

集成说明

- 与现有的 MAP 日志记录（`paklog`）兼容
- 与 MAP 事件共享事件日志数据库
- 使用相同的 SCCP/M3UA 基础设施
- 与 CAMELSessionsLive 一起工作以进行监控
- 与现有的 M3UA 路由集成

修改的文件

- `config/runtime.exs` - 已更新

依赖项

- 现有的 CapRequestGenerator
- 用于 M3UA 发送的 CapClient
- 用于数据包传输的 M3UA.Server
- 用于消息日志记录的 EventLog
- Phoenix LiveView 框架

- 用于 UI 基础设施的控制面板

CAMEL 网关配置□南

概述

CAMEL 网关 **(CAMELGW)** 模式将 OmniSS7 转换为一个智能网络 (IN) 平台，提供基于 CAMEL 应用部分 (CAP) 协议的实时呼叫控制和计费服务。

什么是 CAMEL?

CAMEL (定制的移动网络增强逻辑应用) 是一组标准，旨在在 GSM 核心网络或 UMTS 网络上工作。它允许运营商提供需要实时呼叫控制的服务，例如:

- 预付费呼叫 - 实时余额检查和计费
- 增值服务 - 对增值服务的特殊计费
- 呼叫路由控制 - 基于时间/位置的动态目的地路由
- 虚拟专用网络 - 企业编号计划
- 呼叫筛选 - 根据标准允许/阻止呼叫

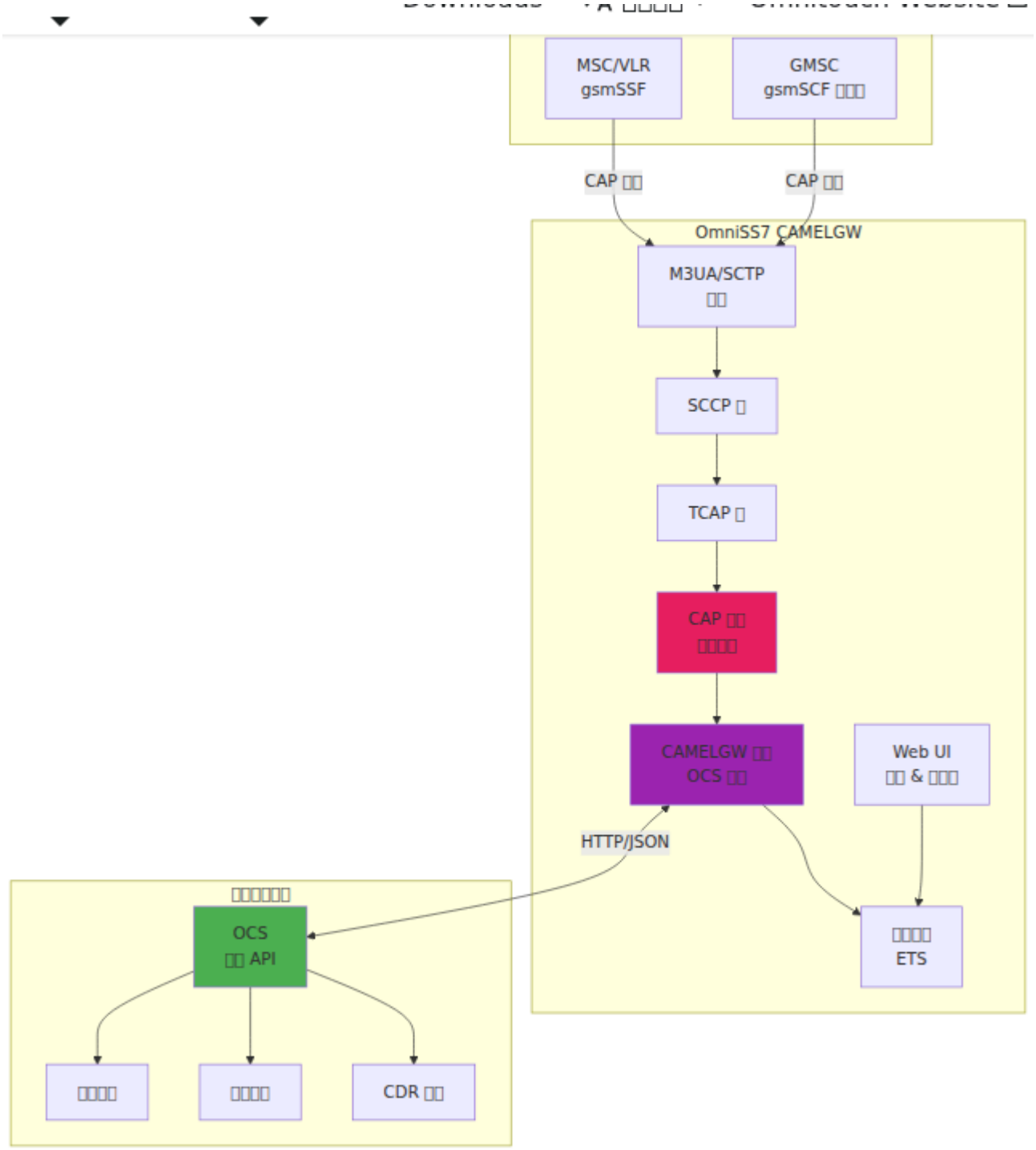
CAP 协议版本

OmniSS7 CAMELGW 支持多个 CAP 版本:

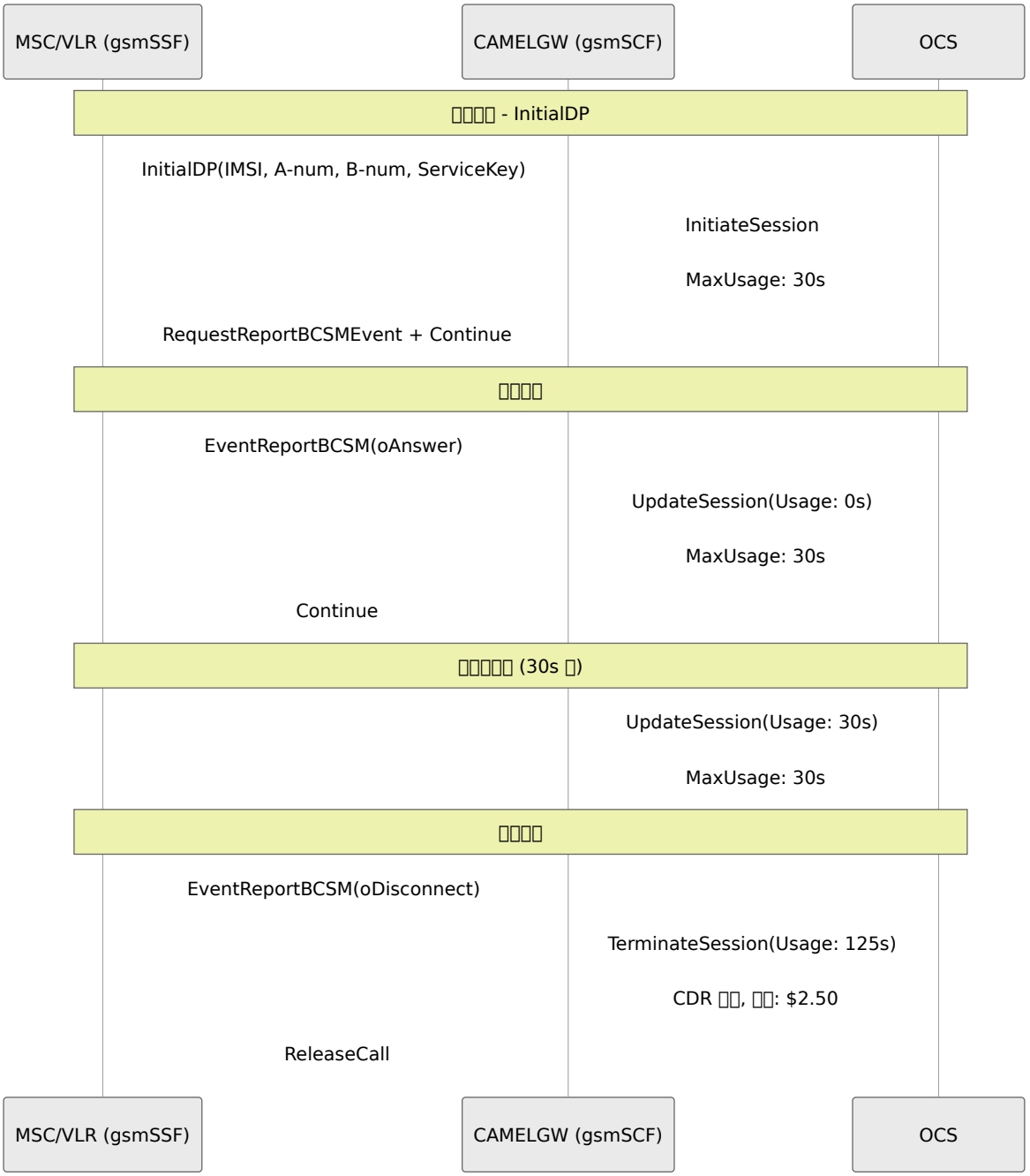
版本	阶段	特性
CAP v1	CAMEL 阶段 1	基本呼叫控制，操作有限
CAP v2	CAMEL 阶段 2	增强操作，支持 SMS
CAP v3	CAMEL 阶段 3	支持 GPRS，额外操作
CAP v4	CAMEL 阶段 4	高级特性，多媒体支持

默认： CAP v2（最广泛部署）

□□


MSC/VLR
gsmSSF
GMSC
gsmSCF □□□
CAP □□
CAP □□
OmniSS7 CAMELGW
M3UA/SCTP
□□
SCCP □
TCAP □
CAP □□
□□□□
CAMELGW □□
OCS □□
Web UI
□□ & □□□
HTTP/JSON
□□□□□□□
OCS
□□ API
□□□□□
ETS
□□□□
□□□□
CDR □□

□□□□□□



□□

□□□□

- □□□□□□ OmniSS7
- □ MSC/GMSC (gsmSSF) □ M3UA □□

- 在线计费系统 (OCS) 具有 API 端点（可选，用于实时计费）

启用 **CAMEL** 网关模式

编辑 `config/runtime.exs` 并配置 CAMEL 网关部分：

```
config :omniss7,
  # 模式标志 - 启用 CAP/CAMEL 特性
  cap_client_enabled: true,
  camelgw_mode_enabled: true,

  # 禁用其他模式
  map_client_enabled: false,
  hlr_mode_enabled: false,
  smsc_mode_enabled: false,

  # CAP/CAMEL 版本配置
  # 确定用于外发请求和对话的 CAP 版本
  # 选项: :v1, :v2, :v3, :v4
  cap_version: :v2,

  # OCS 集成（用于实时计费）
  ocs_enabled: true,
  ocs_url: "http://your-ocs-server/api/charging",
  ocs_timeout: 5000,  # 毫秒
  ocs_auth_token: "your-api-token"  # 可选，如果 OCS 需要身份验证

  # CAMEL 的 M3UA 连接配置
  # 作为 ASP（应用服务器进程）连接以进行 CAP 操作
  cap_client_m3ua: %{
    mode: "ASP",
    callback: {CapClient, :handle_payload, []},
    process_name: :camelgw_client_asp,

    # 本地端点（CAMELGW 系统）
    local_ip: {10, 179, 4, 13},
    local_port: 2905,

    # 远程端点（MSC/GMSC - gsmSSF）
    remote_ip: {10, 179, 4, 10},
    remote_port: 2905,

    # M3UA 参数
    routing_context: 1,
    network_appearance: 0,
    asp_identifier: 13
  }
```

配置 **Web UI** 页面

Web UI 包含用于 CAMEL 操作的专用页面:

```
config :control_panel,
  use_additional_pages: [
    {SS7.Web.EventsLive, "/events", "SS7 事件"},
    {SS7.Web.TestClientLive, "/client", "SS7 客户端"},
    {SS7.Web.M3UAStatusLive, "/m3ua", "M3UA"},
    {SS7.Web.CAMELSessionsLive, "/camel_sessions", "CAP 会话"},
    {SS7.Web.CAMELRequestLive, "/camel_request", "CAP 请求"}
  ],
  page_order: ["/events", "/client", "/m3ua", "/camel_sessions",
               "/camel_request", "/application", "/configuration"]
```

支持的 **CAP** 操作

入站操作（从 **gsmSSF → gsmSCF**）

操作	操作码	描述	处理程序
InitialDP	0	初始检测点 - 呼叫建立通知	`handle_initial_dp/1`
EventReportBCSM	6	基本呼叫状态模型事件（应答、断开等）	`handle_event_report_bcsm/1`
ApplyChargingReport	71	来自 gsmSSF 的计费报告	`handle_apply_charging_rep`
AssistRequestInstructions	16	来自 gsmSRF 的协助请求	`handle_assist_request_ins`

□□□□□□ gsmSCF → gsmSSF□

□□	□□□	□□	□□□
Connect	20	□□□□□□□□□□□□□□	`CapRequestGenerator.connect_req`
Continue	31	□□□□□□□□□□□□□□	`CapRequestGenerator.continue_re`
ReleaseCall	22	□□/□□□□□	`CapRequestGenerator.release_cal`
RequestReportBCSMEvent	23	□□□	`CapRequestGenerator.request_rep`

操作	操作码	描述	生成器
		叫事件通知	
ApplyCharging	35	对呼叫应用计费	`CapRequestGenerator.apply_charg`

Web UI 特性

CAMEL 会话页面

URL: `http://localhost/camel_sessions`

实时监控活动的 CAMEL 呼叫会话:

特性:

- 实时会话列表 - 每 2 秒自动刷新
- 会话详情 - OTID、呼叫 ID、状态、持续时间
- **CAP** 版本 - 显示从 InitialDP 检测到的协议版本 (CAP v1/v2/v3/v4)
- 呼叫信息 - IMSI、A 号码、B 号码、服务密钥
- 状态跟踪 - 已发起、已应答、已终止
- 持续时间计时器 - 实时呼叫持续时间显示

表格列:

- 呼叫 ID、状态、版本、IMSI、主叫号码、被叫号码、服务密钥、持续时间、开始时间、OTID

会话状态：

- □ 已发起 - 收到 InitialDP，等待应答
- □ 已应答 - 呼叫已应答，计费进行中
- □ 已终止 - 呼叫结束，生成 CDR

CAP 版本检测：系统自动从 InitialDP 对话部分检测 CAP 协议版本，并在版本列中显示。这有助于识别每个 MSC 使用的 CAP 版本。

CAMEL 请求构建器

URL: `http://localhost/camel_request`

用于构建和发送 CAP 请求的交互式工具：

特性：

- 请求类型选择器 - InitialDP、Connect、ReleaseCall 等
- 动态表单字段 - 根据选定的请求类型调整
- **SCCP/M3UA** 选项 - 高级寻址配置
- 请求历史 - 最近 20 个请求及其状态
- 会话跟踪 - 维护 OTID 以便后续请求
- 实时反馈 - 成功/错误消息

请求类型：

1. **InitialDP** - 启��新呼叫会话
    - 服务密钥（整数）
    - 主叫号码（A 方）
    - 被叫号码（B 方）
2. **Connect** - 将呼叫路由到目的地
    - 目的号码
3. **ReleaseCall** - 终止呼叫

    - 原因代码（16=正常，17=忙，31=未指定）

4. **RequestReportBCSMEvent** - 请求事件通知
    - 事件：oAnswer、oDisconnect、tAnswer、tDisconnect
5. **Continue** - 继续呼叫而不修改
    - 无需参数
6. **ApplyCharging** - 应用呼叫持续时间限制
    - 持续时间（秒，1-864000）
    - 超时释放（布尔值）
    - 详细用法请参见 CAMEL 请求构建器指南

高级 **SCCP** 选项：

- 被叫方全球标题
- 主叫方全球标题
- 被叫 SSN（默认：146 = gsmSSF）
- 主叫 SSN（默认：146）

M3UA 选项：

- OPC（起始点代码，默认：5013）
- DPC（目的点代码，默认：5011）

与 **OCS** 的集成

呼叫生命周期与计费

1. 呼叫发起 (InitialDP)

当 MSC 发送 InitialDP 时，CAMELGW：

1. 检测 **CAP** 版本 - 检查对话部分以识别 CAP v1/v2/v3/v4
2. 解码 **CAP** 消息 - 提取 IMSI、主叫/被叫号码

3. 调用 **OCS** - `InitiateSession` API
4. 接收授权 - MaxUsage（例如，30 秒）
5. 存储会话 - ��� SessionStore (ETS 表) 中存储 CAP 版本
6. 响应 **MSC** - RequestReportBCSMEvent + Continue（使用相同的 CAP 版本）

示例：

```
# 解码的 InitialDP 数据
%{
  imsi: "310150123456789",
  calling_party_number: "14155551234",
  called_party_number: "14155556789",
  service_key: 1,
  msc_address: "19216800123",
  cap_version: :v2  # 从对话中检测到
}

# OCS 响应
{:ok, %{max_usage: 30}}  # 授权 30 秒

# SessionStore 条目
%{
  call_id: "CAMEL-4B000173",
  initial_dp_data: %{...},
  cap_version: :v2,  # 为响应生成存储
  start_time: 1730246400,
  state: :initiated
}
```

2. 呼叫应答 (EventReportBCSM - oAnswer)

当呼叫被应答时：

1. 接收 **oAnswer** 事件 - 来自 MSC
2. 更新 **OCS** - `UpdateSession`，使用=0
3. 开始扣费循环 - OCS 开始计费
4. 更新会话状态 - 在 SessionStore 中为 `:answered`
5. 继续呼叫 - 向 MSC 发送 Continue

3. 定期更新（可选）

对于长时间呼叫，请求额外信用：

```
# 每 30 秒
OCS.Client.update_session(call_id, %{}, current_usage)
```

如果 MaxUsage 回 0，订阅者没有信用 → 发送 ReleaseCall

4. 呼叫终止 (EventReportBCSM - oDisconnect)

当呼叫结束时：

1. 接收 **oDisconnect** 事件 - 来自 MSC
2. 计算总持续时间 - 从会话开始时间
3. 终止 **OCS** 会话 - `TerminateSession` API
4. 生成 **CDR** - 由 OCS 生成最终成本
5. 清理会话 - 从 SessionStore 中删除
6. 发送 **ReleaseCall** - 确认终止给 MSC

CDR 分析

CDR 由您的 OCS 生成，通常包括：

来自 **CAMEL** 的 **CDR** 字段：

- `Account` - IMSI 或主叫号码
- `Destination` - 被叫方号码
- `OriginID` - 唯一呼叫标识符 (CAMEL-OTID)
- `Usage` - 总呼叫持续时间（秒）
- `Cost` - 计算成本
- `IMSI` - 订阅者 IMSI
- `CallingPartyNumber` - A 方
- `CalledPartyNumber` - B 方
- `MSCAddress` - 服务 MSC 点代码
- `ServiceKey` - CAMEL 服务密钥

测试

使用请求构建器进行手动测试

1. 导航到请求构建器：

   ```
   http://localhost/camel_request
   ```

2. 发送 **InitialDP**：

   - 从下拉菜单中选择 "InitialDP"
   - 服务密钥：`100`
   - 主叫号码：`14155551234`
   - 被叫号码：`14155556789`
   - 点击 "发送 InitialDP 请求"
   - 注意生成的 OTID

3. 监控会话：

   - 打开新标签页：`http://localhost/camel_sessions`
   - 查看状态为 "已发起" 的活动会话

4. 模拟呼叫应答：

   - 回请求构建器
   - 选择 "EventReportBCSM"
   - 事件类型：`oAnswer`
   - 点击 "发送 EventReportBCSM 请求"
   - 会话状态更改为 "已应答"

5. 结束呼叫：

   - 选择 "ReleaseCall"
   - 原因代码：`16`（正常）
   - 点击 "发送 ReleaseCall 请求"
   - 会话状态更改为 "已终止"

使用真实 MSC 进行测试

配置 MSC CAMEL 服务

在您的 MSC/VLR 上，配置 CAMEL 服务：

```
# 示例华为 MSC 配置
ADD CAMELSERVICE:
  SERVICEID=1,
  SERVICEKEY=100,
  GSMSCFADDR="55512341234",  # CAMELGW 全球标题
  DEFAULTCALLHANDLING=CONTINUE;

ADD CAMELSUBSCRIBER:
  IMSI="310150123456789",
  SERVICEID=1,
  TRIGGERTYPE=TERMCALL;
```

监控日志

观察 CAMELGW 日志以获取传入的 CAP 消息：

```
# 实时查看日志
tail -f /var/log/omniss7/omniss7.log

# 过滤 CAP 事件
grep "CAP:" /var/log/omniss7/omniss7.log

# 查看事件日志（JSON 格式）
curl http://localhost/api/events | jq '.[] | select(.map_event |
startswith("CAP:"))'
```

负载测试

在循环中使用请求构建器进行负载测试：

```
# 发送 100 个 InitialDP 请求
for i in {1..100}; do
  curl -X POST http://localhost/api/camel/initial_dp \
    -H "Content-Type: application/json" \
    -d '{
      "service_key": 100,
      "calling_number": "1415555'$i'",
      "called_number": "14155556789"
    }'
  sleep 0.1
done
```

监控与操作

Prometheus 指标

CAMELGW 在 `http://localhost:8080/metrics` 处公开指标:

特定于 **CAP** 的指标:

- `cap_requests_total{operation}` - 按操作类型（例如，initialDP、requestReportBCSMEvent）统计的总 CAP 请求

其他 **MAP/API** 指标:

- `map_requests_total{operation}` - 按操作类型统计的总 MAP 请求
- `map_request_duration_milliseconds{operation}` - 请求持续时间直方图
- `map_pending_requests` - 待处理 MAP 事务的数量

M3UA STP 指标（如果启用 STP 模式）:

- `m3ua_stp_messages_received_total{peer_name,point_code}` - 从对等方接收到的消息
- `m3ua_stp_messages_sent_total{peer_name,point_code}` - 发送到对等方的消息
- `m3ua_stp_routing_failures_total{reason}` - 按原因统计的路由失败

示例查询:

```
# CAP 请求
curl http://localhost:8080/metrics | grep cap_requests_total

# 收到的总 InitialDP
curl http://localhost:8080/metrics | grep
'cap_requests_total{operation="initialDP"}'

# MAP 待处理请求
curl http://localhost:8080/metrics | grep map_pending_requests
```

健康检查

```
# 检查 M3UA 连接
curl http://localhost/api/m3ua-status

# 检查 OCS 连接
curl http://localhost/api/ocs-status

# 检查活动会话
curl http://localhost/api/camel/sessions/count
```

日志配置

在 `config/runtime.exs` 中调整日志级别:

```
config :logger,
  level: :info  # 选项: :debug, :info, :warning, :error

# 启用 CAP 调试日志
config :logger, :console,
  metadata: [:cap_operation, :otid, :call_id]
```

故障排除

问题：未接收到 **CAP** 消息

症状：请求构建器正常工作，但 MSC 不发送 InitialDP

检查：

1. M3UA 链接状态：`curl http://localhost/api/m3ua-status`
2. MSC CAMEL 服务配置（服务密钥，gsmSCF 地址）
3. SCCP 路由（全球标题必须路由到 CAMELGW）
4. 防火墙规则（允许 SCTP 端口 2905）

解决方案：

```
# 验证 M3UA 连接
tcpdump -i eth0 sctp

# 检查 MSC 是否能够到达 CAMELGW
ss -tuln | grep 2905
```

问题：**OCS** 错误

症状：`INSUFFICIENT_CREDIT` 或超时错误

检查：

1. OCS 可达：`curl http://your-ocs-server/api/health`
2. 账户在 OCS 中有余额
3. OCS 中配置了计费计划
4. 与 OCS 的网络连接
5. 身份验证令牌有效（如需要）

解决方案：

- 验证 `runtime.exs` 中的 OCS URL 配置
- 检查 OCS 日志中的错误

- 使用 curl 手动测试 OCS API
- 验证防火墙规则允许连接

问题：未找到会话

症状：EventReportBCSM 失败，提示 "未找到会话"

原因：OTID 不匹配或会话过期

解决方案：

1. 验证日志中的 OTID
2. 检查会话超时（默认：无过期）
3. 确保 DTID 在 Continue/End 消息中与 OTID 匹配

```
# 检查活动会话
iex> CAMELGW.SessionStore.list_sessions()
```

问题：解码错误

症状：日志中出现 `Failed to decode InitialDP`

原因：CAP 版本不匹配或消息格式错误

解决方案：

1. 检查 CAP 版本配置是否与 MSC 匹配
2. 验证 ASN.1 编码是否正确
3. 捕获 PCAP 并使用 Wireshark 分析

```
# 捕获 CAP 消息
tcpdump -i eth0 -w cap_trace.pcap sctp port 2905

# 使用 Wireshark 分析（过滤：m3ua）
wireshark cap_trace.pcap
```

高级配置

多个 CAP 版本

支持每个服务密钥不同的 CAP 版本:

```
config :omniss7,
  cap_version_map: %{
    100 => :v2,  # 服务密钥 100 使用 CAP v2
    200 => :v3,  # 服务密钥 200 使用 CAP v3
    300 => :v4   # 服务密钥 300 使用 CAP v4
  },
  cap_version: :v2  # 默认
```

总结

CAMEL 网关模式使 OmniSS7 能够作为一个完整的智能网络平台，具有:

□ **□□□ CAP** 协议支持 (v1/v2/v3/v4)
□ **□□ OCS** 集成的实时计费
□ □□□□□□ (Connect, Release, Continue)
□ **□□ ETS** 存储的会话管理
□ **□□ Web UI** 请求构建器的交互式测试
□ □□□□□□□□□□□
□ □□□□□□□□ **CDR** 生成
□ □□□□□□□□□□□

有关更多信息:

- CAMEL 请求构建器文档
- 技术参考 - CAP 操作

产品: OmniSS7 CAMEL 网关
文档版本: 1.0
最后更新: 2025-10-26

常见功能□南

← □回主文档

本指南涵盖所有 OmniSS7 操作模式的常见功能。

目录

Web UI 概述

Web UI 可通过您配置的 Web 服务器地址访问。

主要导航

- 事件 - 实时 SS7 信令事件和消息日志
- 应用程序 - 应用程序状态和运行时信息
- 配置 - 系统配置查看器
- **M3UA** 状态 - M3UA 对等连接（STP 模式）
- 短信队列 - 外发短信消息（SMSc 模式）

访问 Web UI

1. 打开您的 Web 浏览器
2. 导航到配置的主机名（例如，`http://localhost`）
3. 查看系统状态仪表板

Swagger API 文档

交互式 API 文档:

```
http://your-server/swagger
```

Web UI 配置

在 `config/runtime.exs` 中配置:

```
config :control_panel,
  # 导航菜单中的页面顺序
  page_order: ["/events", "/application", "/configuration"],

  # Web 服务器配置
  web: %{
    listen_ip: "0.0.0.0",     # 绑定的 IP (0.0.0.0 表示所有接口)
    port: 80,                  # HTTP 端口 (HTTPS 使用 443)
    hostname: "localhost",     # URL 生成的服务器主机名
    enable_tls: false,         # 配置为 true 以启用 HTTPS
    tls_cert: "cert.pem",     # TLS 证书文件路径
    tls_key: "key.pem"        # TLS 私钥文件路径
  }
```

配置参数:

参数	类型	默认值	描述
`page_order`	列表	`["/events", "/application", "/configuration"]`	导航菜单中的页面顺序
`listen_ip`	字符串	`"0.0.0.0"`	绑定 Web 服务器的 IP 地址
`port`	整数	`80`	HTTP 端口（HTTPS 使用 443）
`hostname`	字符串	`"localhost"`	URL 生成的服务器主机名
`enable_tls`	布尔值	`false`	启用 TLS 的 HTTPS
`tls_cert`	字符串	`"cert.pem"`	TLS 证书路径（启用 TLS 时）
`tls_key`	字符串	`"key.pem"`	TLS 私钥路径（启用 TLS 时）

日志配置

在 `config/runtime.exs` 中配置日志级别：

```
config :logger,
  level: :debug  # 选项: :debug, :info, :warning, :error
```

日志级别：

- `:debug` - 详细调试信息
- `:info` - 一般信息消息
- `:warning` - 潜在问题的警告消息
- `:error` - 仅错误消息

API 文档

API 基础 URL

```
http://your-server/api
```

响应代码

- **200** - 成功
- **400** - 错误请求
- **504** - 网关超时

OpenAPI 规范

```
http://your-server/swagger.json
```

监控与□标

Prometheus □标端点

```
http://your-server/metrics
```

关键□标类别

M3UA/SCTP □标:

- SCTP 关联状态变化
- M3UA ASP 状态转换
- 发送/接收的协议数据单元

M2PA 指标:

- 链路状态转换（DOWN → ALIGNMENT → PROVING → READY）
- 每个链路发送/接收的消息和字节
- 链路特定错误（解码、编码、SCTP）

STP 指标:

- 每个对等体接收/发送的消息
- 按原因的路由失败
- 在对等体之间的流量分布

MAP 客户端指标:

- 按操作类型的 MAP 请求
- 请求持续时间直方图
- 待处理事务计量

CAP 指标:

- 按操作类型的 CAP 请求
- CAMEL 网关操作

SMSc 指标:

- 队列深度
- 投递率
- 失败的消息

Grafana 集成

OmniSS7 指标与 Prometheus 和 Grafana 兼容。

最佳实践

安全建议

1. 网络隔离
   - 部署在专用 VLAN 中
   - 防火墙规则限制访问
   - 仅允许来自已知地址的 SCTP
2. **Web UI** 安全
   - 在生产环境中启用 TLS
   - 使用带身份验证的反向代理
   - 限制为管理 IP
3. **API** 安全
   - 实施速率限制
   - 使用 API 密钥或 OAuth
   - 记录所有请求以便审计

性能调优

1. **TPS** 限制
   - 配置适当的 TPS
   - 监控系统负载
   - 调整 SCTP 缓冲区
2. 数据库优化
   - 添加索引
   - 归档旧消息
   - 监控连接池
3. **M3UA** 调优

  - 调整 SCTP 心跳间隔
  - 配置超时值
  - 使用多个链路以实现冗余

SCTP 多宿主网络冗余

什么是 **SCTP** 多宿主?

SCTP 多宿主是 SCTP 协议的内置功能，允许单个 M3UA 连接绑定到同一网络接口或不同网络接口上的多个 IP 地址。这提供了传输层的自动故障转移和冗余。

主要好处：

- 自动故障转移：如果一条网络路径失败，SCTP 会自动切换到备用路径而不丢失连接
- 零配置故障转移：无需应用级逻辑 - SCTP 处理路径监控和故障转移
- 提高可靠性：能够在网络故障、交换机故障或网卡故障中生存
- 负载均衡：SCTP 可以在多个路径之间分配流量（依赖于实现）

工作原理

当您为 M3UA 连接配置多个 IP 地址时，SCTP:

1. 绑定到所有 **IP**：套接字同时绑定到所有配置的 IP 地址
2. 监控路径：SCTP 持续在所有路径上发送心跳数据包以监控其健康状况
3. 检测故障：如果主路径的心跳失败，SCTP 将其标记为不可达
4. 自动故障转移：流量立即切换到备用路径，无需应用干预
5. 路径恢复：当故障路径恢复时，SCTP 检测到并再次标记为可用

配置

SCTP 多宿主通过提供 **IP** 地址列表 而不是单个 IP 元组进行配置。

单个 **IP**（传统）

```
# 单个 IP - 无多宿主
local_ip: {10, 179, 4, 10}
```

多个 **IP**（启用多宿主）

```
# 多个 IP - 启用多宿主
# 第一个 IP 为主，后续 IP 为备用路径
local_ip: [{10, 179, 4, 10}, {10, 179, 4, 11}]
```

配置示例

示例 **1**：具有多宿主的 **STP** 对等体

```
# STP 模式对等体配置
config :omniss7,
  m3ua_peers: [
    %{
      peer_id: 1,
      name: "Partner_STP_Redundant",
      role: :client,
      # 多宿主：绑定两个本地 IP 以实现冗余
      local_ip: [{213, 57, 23, 200}, {213, 57, 23, 201}],
      local_port: 0,
      # 远程对等体也支持多宿主
      remote_ip: [{213, 57, 23, 100}, {213, 57, 23, 101}],
      remote_port: 2905,
      routing_context: 1,
      point_code: 100,
      network_indicator: :international
    }
  ]
```

示例 **2**：具有多宿主的 **MAP** 客户端

```
# MAP 客户端模式与多宿主
config :omniss7,
  map_client_enabled: true,
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :hlr_client_asp,
    # 多宿主: 两个本地 IP 以实现故障转移
    local_ip: [{10, 0, 0, 100}, {10, 0, 0, 101}],
    local_port: 2905,
    # 远程 STP 支持多宿主
    remote_ip: [{10, 0, 0, 1}, {10, 0, 0, 2}],
    remote_port: 2905,
    routing_context: 1
  }
```

示例 **3**: 具有多宿主的 **STP** 监听器

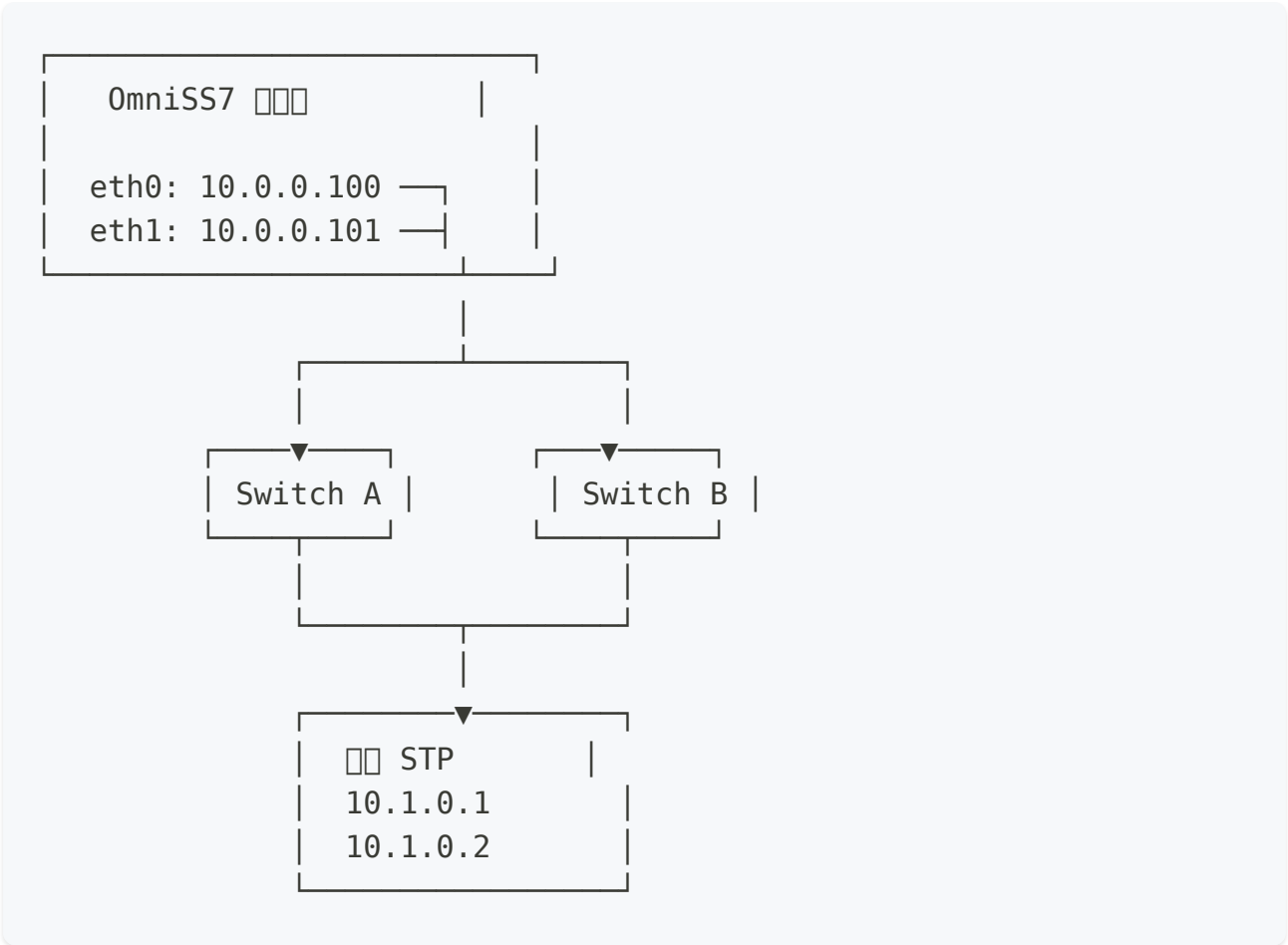
```
# 独立的 STP 服务器与多宿主
config :omniss7,
  m3ua_stp: %{
    enabled: true,
    # 监听多个 IP 以接收传入连接
    local_ip: [{172, 16, 0, 10}, {172, 16, 0, 11}],
    local_port: 2905,
    point_code: 100
  }
```

示例 **4**: 混合配置（向后兼容）

```
# 单个和多宿主对等体的混合
config :omniss7,
  m3ua_peers: [
    # 传统对等体 - 单个 IP
    %{
      peer_id: 1,
      name: "Legacy_STP",
      role: :client,
      local_ip: {10, 0, 0, 1},       # 单个 IP 元组
      local_port: 0,
      remote_ip: {10, 0, 0, 10},
      remote_port: 2905,
      routing_context: 1,
      point_code: 100
    },
    # 新对等体 - 多宿主
    %{
      peer_id: 2,
      name: "Redundant_STP",
      role: :client,
      local_ip: [{10, 0, 0, 2}, {10, 0, 0, 3}],  # IP 列表
      local_port: 0,
      remote_ip: [{10, 0, 0, 20}, {10, 0, 0, 21}],
      remote_port: 2905,
      routing_context: 2,
      point_code: 200
    }
  ]
```

网络拓扑场景

场景 **1**：双 **NIC**（常见部署）



```
┌──────────────────────────┐
│   OmniSS7 服务器        │
│                          │
│  eth0: 10.0.0.100 ──┐    │
│  eth1: 10.0.0.101 ──┤    │
└──────────────────────┼────┘
                       │
              ┌────────┴────────┐
              │                 │
         ┌────▼────┐       ┌────▼────┐
         │ Switch A │      │ Switch B │
         └────┬────┘       └────┬────┘
              │                 │
              └────────┬────────┘
                       │
              ┌────────▼────────┐
              │  远程 STP       │
              │  10.1.0.1       │
              │  10.1.0.2       │
              └─────────────────┘
```

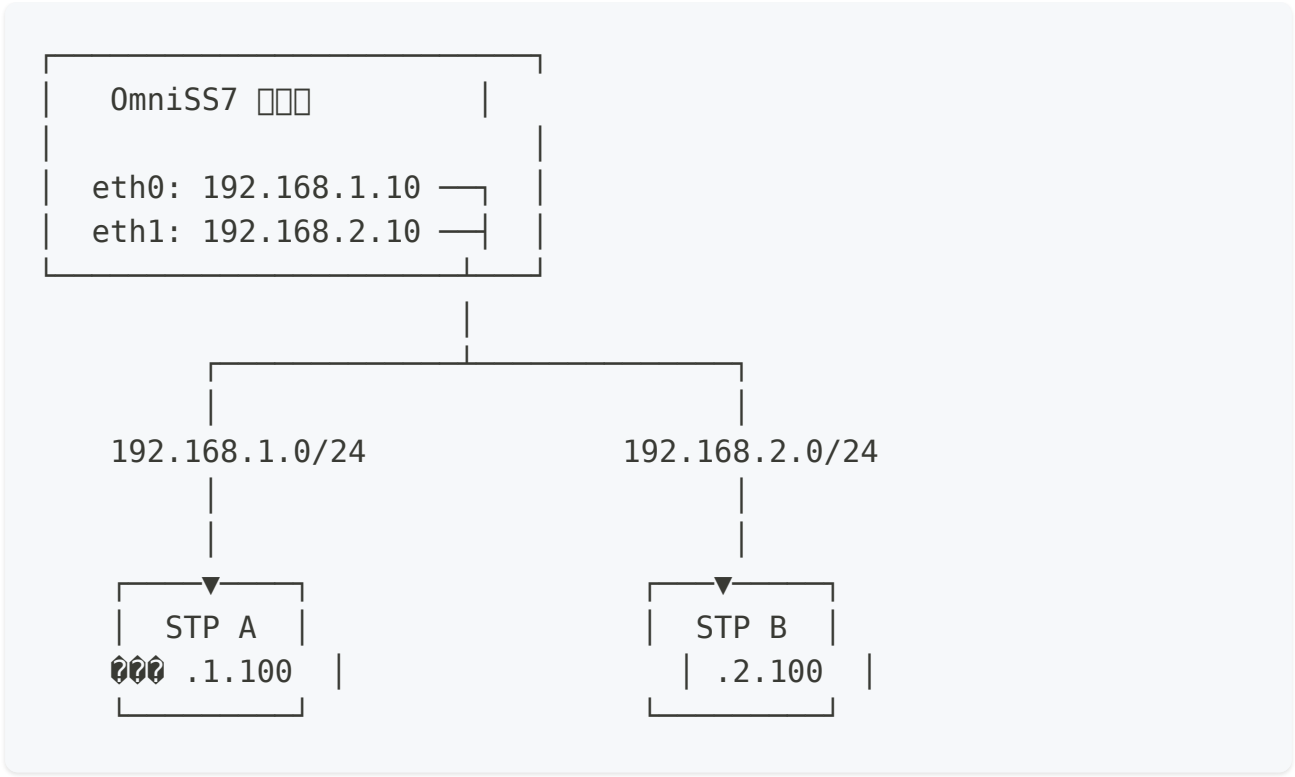
配置:

```
local_ip: [{10, 0, 0, 100}, {10, 0, 0, 101}]   # 两个 NIC
remote_ip: [{10, 1, 0, 1}, {10, 1, 0, 2}]      # 远程对等体
```

好处:

- 在一个 NIC 故障时仍能存活
- 在一个交换机故障时仍能存活
- 自动故障转移小于 1 秒

场景 2: 多个子网



□□□

```
local_ip: [{192, 168, 1, 10}, {192, 168, 2, 10}]
remote_ip: [{192, 168, 1, 100}, {192, 168, 2, 100}]
```

□□□

- □□□□□□□□□□□□
- □□□□□□□□□□
- □□□□□□□□□

□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□

```
[info] SCTP □□□□□□□□□□□ 2 □□□□ IP
[info] STP □□□□□□□□□□□□□□ 2 □□□□ IP
```

□□□□□□□□□□

```
[warning] [MULTIHOMING] 路径 10.0.0.100 对于对等体 Partner_STP (assoc_id=1) 不可达
[info] [MULTIHOMING] 路径 10.0.0.101 现在是对等体 Partner_STP (assoc_id=1) 的主路径
[info] [MULTIHOMING] 路径 10.0.0.100 现在对等体 Partner_STP (assoc_id=1) 可用
```

Web UI 显示

Web UI 自动显示多宿主信息：

M3UA 状态页面：

- 单个 **IP**：显示为 `10.0.0.100`
- 多个 **IP**：显示为 `10.0.0.100 (+1)` 或 `10.0.0.100 (+2)`
- 详细视图：显示所有 IP 及主/备用标签

最佳实践

1. 网络设计

- 使用不同的 **NIC** 以实现最大冗余
- 不同的交换机 以存活交换机故障
- 不同的子网 如果可能以实现路由多样性
- 同一数据中心初期 - 测试后再进行地理分离

2. IP 地址规划

- 第一个 **IP** 是主 - 确保它在最可靠的路径上
- 顺序很重要 - 按优先顺序列出 IP
- 一致的寻址 - 使用相似的寻址方案以便于故障排除

3. 测试故障转移

```
# 禁用主接口以测试故障转移
sudo ip link set eth0 down

# 监控日志以查看故障转移
tail -f /var/log/omniss7.log | grep MULTIHOMING

# 重新启用接口
sudo ip link set eth0 up
```

4. 双方应支持多宿主

- 最佳：本地和远程都使用多个 IP
- 可接受：仅一方使用多宿主
- 注意：当两个端点都支持时冗余效果最佳

5. 防火墙配置

```
# 允许所有多宿主 IP 上的 SCTP
iptables -A INPUT -p sctp --dport 2905 -s 10.0.0.0/24 -j ACCEPT
iptables -A INPUT -p sctp --dport 2905 -s 10.1.0.0/24 -j ACCEPT
```

故障排除

问题：多宿主无法工作

症状：仅使用主 IP，无故障转移

检查：

1. 验证 Erlang SCTP 支持：`erl -eval 'gen_sctp:open(9999, [binary, {ip, {127,0,0,1}}]).'`
2. 检查内核 SCTP 模块：`lsmod | grep sctp`
3. 如有需要加载 SCTP：`sudo modprobe sctp`
4. 验证系统上配置了两个 IP：`ip addr show`

问题：路径未故障转移

症状：主路径标记为关闭但流量未切换

检查：

1. 检查 SCTP 心跳设置
2. 验证路由表中有所有路径的路由
3. 检查防火墙允许所有 IP 上的 SCTP
4. 查看 SCTP 路径监控日志

问题：频繁的路径波动

症状：路径在 UP 和 DOWN 之间不断切换

检查：

1. 网络不稳定 - 检查物理链接
2. SCTP 心跳过于激进 - 可能需要调优
3. 防火墙丢弃 SCTP 心跳
4. 一条路径上的 MTU 问题

性能考虑

- 最小开销：SCTP 心跳小且不频繁
- 无需应用更改：多宿主对应用层透明
- 快速故障转移：通常小于 1 秒检测和故障转移
- 自动恢复：无需手动干预

兼容性

- 向后兼容：单个 IP 元组格式仍然有效
- 混合部署：可以混合单 IP 和多 IP 对等体
- 支持所有模式：在 STP、HLR、SMSc 和 MAP 客户端模式下均可工作
- **Erlang** 要求：需要编译支持 SCTP 的 Erlang

监控与警报

关键□标：

- M3UA 连接状态

- MAP □□□□□
- API □□□□
- □□□□□□□

□□□□□

- M3UA □□ 1 □□
- MAP □□□□□ 10%
- □□□□□□ 1000
- API □□□□□ 5%

□□□□□□

□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□ (`:logger`)

```
config :logger,
  level: :debug  # :debug | :info | :warning | :error
```

Web UI □□ (`:control_panel`)

```
config :control_panel,
  page_order: ["/events", "/application", "/configuration"],
  web: %{
    listen_ip: "0.0.0.0",
    port: 80,
    hostname: "localhost",
    enable_tls: false,
    tls_cert: "cert.pem",
    tls_key: "key.pem"
  }
```

□□	□□	□□	□□□	□□
`page_order`	□□□□□□	□	`["/events", "/application", "/configuration"]`	□□□□□□□□□
`web.listen_ip`	□□□	□	`"0.0.0.0"`	□□ Web □□□□ IP □□
`web.port`	□□	□	`80`	HTTP/HTTPS □□□
`web.hostname`	□□□	□	`"localhost"`	□□□□□□
`web.enable_tls`	□□□	□	`false`	□□ HTTPS
`web.tls_cert`	□□□	□□□□ TLS	`"cert.pem"`	TLS □□□□
`web.tls_key`	□□□	□□□□ TLS	`"key.pem"`	TLS □□□□

M3UA STP □□ (`:omniss7`)

```
config :omniss7,
  m3ua_stp: %{
    enabled: false,
    local_ip: {127, 0, 0, 1},
    local_port: 2905
  },
  enable_gt_routing: true,
  m3ua_peers: [...],
  m3ua_routes: [...],
  m3ua_gt_routes: [...]
```

参数	类型	必需	默认值	描述
`m3ua_stp.enabled`	布尔值	是	`false`	启动��启用 STP 模式
`m3ua_stp.local_ip`	元组	是	`{127, 0, 0, 1}`	绑定用于接收 M3UA 的 IP
`m3ua_stp.local_port`	整数	是	`2905`	M3UA 的 SCTP 端口
`enable_gt_routing`	布尔值	否	`false`	启用全球标题路由

M3UA 对等体参数:

参数	类型	必需	描述
`peer_id`	整数	是	唯一对等体标识符
`name`	字符串	是	描述性对等体名称
`role`	原子	是	`:client` 或 `:server`
`local_ip`	元组或列表	如果是 `:client`	本地 IP（单个：`{10, 0, 0, 1}` 或列表：`[{10, 0, 0, 1}, {10, 0, 0, 2}]`）
`local_port`	整数	如果是 `:client`	本地端口（0 表示动态）
`remote_ip`	元组或列表	是	远程对等体 IP（单个：`{10, 0, 0, 10}` 或列表：`[{10, 0, 0, 10}, {10, 0, 0, 11}]`）
`remote_port`	整数	如果是 `:client`	远程对等体端口
`routing_context`	整数	是	M3UA 路由上下文
`point_code`	整数	是	SS7 点代码
`network_indicator`	原子	否	`:international` 或 `:national`

M3UA 路由参数：

□□	□□	□□	□□
`dest_pc`	□□	□	□□□□□
`peer_id`	□□	◆◆	□□□□□□□□□
`priority`	□□	□	□□□□□□□□□□□□□□□□□□□□□□□
`network_indicator`	□□	□	`:international` □ `:national`

M3UA GT □□□□□

□□	□□	□□	□□
`gt_prefix`	□□□	□	□□□□□□□□□□
`peer_id`	□□	□	□□□□□
`priority`	□□	□	□□□□□
`description`	□□□	□	□□□□□□□□□
`source_ssn`	□□	□	□□□ SSN □□□□□
`dest_ssn`	□□	□	□□□ SSN □□□□□

MAP □□□□□ (`:omniss7`)

```
config :omniss7,
  map_client_enabled: false,
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :map_client_asp,
    local_ip: {10, 0, 0, 100},
    local_port: 2905,
    remote_ip: {10, 0, 0, 1},
    remote_port: 2905,
    routing_context: 1
  }
```

□□	□□	□□	□□□	□□
`map_client_enabled`	□□□□	□	`false`	□□ MAP □□□□□
`map_client_m3ua.mode`	□□□□	□	`"ASP"`	M3UA □□□□ □`"ASP"` □ `"SGP"`□
`map_client_m3ua.callback`	□□	□	`{MapClient, :handle_payload, []}`	□□□□□□ □□
`map_client_m3ua.process_name`	□□	□	`:map_client_asp`	□□□□□□ □
`map_client_m3ua.local_ip`	□□	□	-	□□ IP □□
`map_client_m3ua.local_port`	□□	□	`2905`	□□ SCTP □□
`map_client_m3ua.remote_ip`	□□	□	-	□□ STP/SGP IP
`map_client_m3ua.remote_port`	□□	□	`2905`	□□ SCTP □□
`map_client_m3ua.routing_context`	□□	□	-	M3UA □ □□□□

短信中心配置 (`:omniss7`)

```
config :omniss7,
  auto_flush_enabled: false,
  auto_flush_interval: 10_000,
  auto_flush_dest_smsc: nil,
  auto_flush_tps: 10
```

参数	类型	必需	默认值	描述
`auto_flush_enabled`	布尔值	否	`false`	启用 SMS 队列的自动刷新
`auto_flush_interval`	整数	否	`10000`	队列轮询间隔（毫秒）
`auto_flush_dest_smsc`	字符串/nil	否	`nil`	按目标 SMSC 过滤（nil = 全部）
`auto_flush_tps`	整数	否	`10`	每秒最大事务数

HTTP API 配置 (`:omniss7`)

短信后端现在使用 HTTP API 而不是直接数据库连接。

```
config :omniss7,
  smsc_api_base_url: "https://10.5.198.200:8443",
  frontend_name: "omni-smsc01"  # 可选：默认为 hostname_SMSc
```

API 参数:

参数	类型	必需	默认值	描述
`smsc_api_base_url`	字符串	是	`"https://10.5.198.200:8443"`	短信后端 API 的基础 URL
`frontend_name`	字符串	否	`"{hostname}_SMSc"`	注册的前端标识符

使用的 **API** 端点:

- `POST /api/frontends` - 将此前端实例注册到后端
- `POST /api/messages_raw` - 插入新 SMS 消息
- `GET /api/messages` - 检索消息队列（带 `smsc` 头）
- `PATCH /api/messages/{id}` - 将消息标记为已投递
- `PUT /api/messages/{id}` - 更新消息状态
- `POST /api/events` - 添加事件跟踪
- `GET /api/status` - 健康检查端点

前端注册:

系统在启动时自动向后端 API 注册，并每 5 分钟重新注册。注册内容包括:

- 前端名称和类型（SMSc）
- 主机名
- 运行时间（秒）
- 配置详情（JSON 格式）

配置说明:

- 默认情况下，针对自签名证书禁用 SSL 验证
- HTTP 请求在 5 秒后超时
- 所有时间戳均为 ISO 8601 格式
- API 使用 JSON 作为请求/响应主体

相关文档

- ← 回主文档
- STP 指南
- MAP 客户端指南
- 短信中心指南
- HLR 指南

OmniSS7 由 Omnitouch 网络服务提供

配置参考

← 回主文档

本文档提供了所有 OmniSS7 配置参数的全面参考。

目录

概述

OmniSS7 配置通过 `config/runtime.exs` 管理。系统支持三种操作模式:

- **STP** 模式 - 信号传输点用于路由
- **HLR** 模式 - 用户位置寄存器用于用户管理
- **SMSc** 模式 - 短信中心用于消息传递

配置文件: `config/runtime.exs`

操作模式标志

控制启用哪些功能。

参数	类型	默认	描述	模式
`map_client_enabled`	布尔	`false`	启用 MAP 客户端和 M3UA 连接	所有
`hlr_mode_enabled`	布尔	`false`	启用 HLR 特定功能	HLR
`smsc_mode_enabled`	布尔	`false`	启用 SMSc 特定功能	SMSc

示例：

```
config :omniss7,
  map_client_enabled: true,
  hlr_mode_enabled: true,
  smsc_mode_enabled: false
```

HLR 模式参数

HLR（用户位置寄存器）模式的配置。

HLR API □□

□□	□□	□□	□□	□□
`hlr_api_base_url`	□□□□	-	□	□□ HLR API □□ URL□SSL □□□□□□□□□□
`hlr_service_center_gt_address`	□□□□	-	□	□ UpdateLocation □□□□□□ □ HLR □□□□□□□
`smsc_service_center_gt_address`	□□□□	-	□	□ SRI-for-SM □□□□□□□ SMSC GT □□

□□:

```
config :omniss7,
  hlr_api_base_url: "https://10.180.2.140:8443",
  hlr_service_center_gt_address: "55512341111",
  smsc_service_center_gt_address: "55512341112"
```

MSISDN ↔ IMSI □□

□ MSISDN □□□□ IMSI □□□□□□□□□□□□□□□□□□□□□□□□ HLR □□□□ MSISDN ↔ IMSI □□□

参数	类型	默认	必需	描述
`hlr_imsi_plmn_prefix`	字符串	`"50557"`	否	合成 IMSI 生成的 PLMN 前缀（MCC+MNC）
`hlr_msisdn_country_code`	字符串	`"61"`	否	IMSI→MSISDN 反向映射的国家代码前缀
`hlr_msisdn_nsn_offset`	整数	`0`	否	MSISDN 中 NSN 开始的偏移量（通常是国家代码的长度）
`hlr_msisdn_nsn_length`	整数	`9`	否	从 MSISDN 中提取的国家用户号码的长度

示例（2 位国家代码）：

```
config :omniss7,
  hlr_imsi_plmn_prefix: "50557",       # MCC 505 + MNC 57
  hlr_msisdn_country_code: "99",       # 示例 2 位国家代码
  hlr_msisdn_nsn_offset: 2,            # 跳过 2 位国家代码
  hlr_msisdn_nsn_length: 9             # 提取 9 位 NSN
```

示例（3 位国家代码）：

```
config :omniss7,
  hlr_imsi_plmn_prefix: "50557",       # MCC 505 + MNC 57
  hlr_msisdn_country_code: "999",      # 示例 3 位国家代码
  hlr_msisdn_nsn_offset: 3,            # 跳过 3 位国家代码
  hlr_msisdn_nsn_length: 8             # 提取 8 位 NSN
```

重要：将 `nsn_offset` 设置为国家代码的长度以正确提取 NSN。例如：

- 国家代码 "9"（1 位）→ `nsn_offset: 1`
- 国家代码 "99"（2 位）→ `nsn_offset: 2`

- 国家代码 "999"（3 位）→ `nsn_offset: 3`

InsertSubscriberData (ISD) 配置

在 UpdateLocation 期间发送到 VLR 的用户配置数据的配置。有关 ISD 序列和消息流的详细说明，请参见 HLR 指南中的 InsertSubscriberData 配置。

参数	类型	默认	必需	
`isd_network_access_mode`	原子	`:packetAndCircuit`	否	网络接入类型：`:packetAndCircu` 或 `:circuitOnly`
`isd_send_ss_data`	布尔	`true`	否	发送带有补充服务数据的 ISD
`isd_send_call_barring`	布尔	`true`	否	发送带有呼叫限制数据的 ISD

示例：

```
config :omniss7,
  isd_network_access_mode: :packetAndCircuit,
  isd_send_ss_data: true,
  isd_send_call_barring: true
```

CAMEL 配置

基于 CAMEL 的智能呼叫路由的配置。有关 CAMEL 集成和服务密钥的详细说明，请参见 HLR 指南中的 CAMEL 集成。

参数	类型	默认	必需	
`camel_service_key`	整数	`11_110`	否	用于 S
`camel_trigger_detection_point`	原子	`:termAttemptAuthorized`	否	CAM `:ter`
`camel_gsmscf_gt_address`	字符串	(使用被叫 GT)	否	CAM

示例：

```
config :omniss7,
  camel_service_key: 11_110,
  camel_trigger_detection_point: :termAttemptAuthorized
```

家庭 VLR 前缀

用于区分家庭用户与漫游用户的配置。有关家庭/漫游检测和 PRN 操作的详细说明，请参见 HLR 指南中的漫游用户处理。

参数	类型	默认	必需	描述
`home_vlr_prefixes`	列表	`["5551231"]`	否	被视为“家庭”网络的 VLR GT 前缀

示例：

```
config :omniss7,
  home_vlr_prefixes: ["5551231", "5551234"]
```

SMSc □□□□

SMS □□□□□□□□□

SMSc API □□

□□	□□	□□	□□	□□
`smsc_api_base_url`	□□□□	-	□	□□ SMSc API □□ URL □SSL □□ □□□□□□ □□
`smsc_name`	□□□□	`"{hostname}_SMSc"`	□	□□□□□ SMSc □□
`smsc_service_center_gt_address`	□□□□	-	□	□□□□□□ □□□□

□□:

```
config :omniss7,
  smsc_api_base_url: "https://10.179.3.219:8443",
  smsc_name: "ipsmgw",
  smsc_service_center_gt_address: "55512341112"
```

□□: □□□□□ **5** □□ □□□□□□□□□□□□ `SMS.FrontendRegistry` □□□

自动刷新配置

参数	类型	默认	必需	描述
`auto_flush_enabled`	布尔	`true`	否	启用自动 SMS 队列处理
`auto_flush_interval`	整数	`10_000`	否	队列处理间隔（毫秒）
`auto_flush_dest_smsc`	字符串	-	是	自动刷新目标 SMSC 名称
`auto_flush_tps`	整数	`10`	否	消息处理速率（事务/秒）

示例：

```
config :omniss7,
  auto_flush_enabled: true,
  auto_flush_interval: 10_000,
  auto_flush_dest_smsc: "ipsmgw",
  auto_flush_tps: 10
```

STP 模式参数

M3UA 信号传输点模式的配置。有关详细的路由配置和示例，请参见 STP 配置指南。

□□ STP □□□

□□	□□	□□	□□	□□
`m3ua_stp.enabled`	□□	`false`	□	□□□□ M3UA STP □□□
`m3ua_stp.local_ip`	□□□□□□	`{127, 0, 0, 1}`	□	□□□□□ IP □□□□□ IP: `{10, 0, 0, 1}` □□□ IP □□ SCTP □□□: `[{10, 0, 0, 1}, {10, 0, 0, 2}]`
`m3ua_stp.local_port`	□□	`2905`	□	□□□□□
`m3ua_stp.point_code`	□□	-	□□□□□	□ STP □ SS7 □□□

□□□□□ **IP□**:

```
config :omniss7,
  m3ua_stp: %{
    enabled: true,
    local_ip: {10, 179, 4, 10},
    local_port: 2905,
    point_code: 100
  }
```

□□□**SCTP** □□□□:

```
config :omniss7,
  m3ua_stp: %{
    enabled: true,
    # 多个 IP 用于冗余
    local_ip: [{10, 179, 4, 10}, {10, 179, 4, 11}],
    local_port: 2905,
    point_code: 100
  }
```

> 注意：有关 SCTP 多宿主配置和好处的详细信息，请参见 常见指南中的 SCTP 多宿主。

全局标题路由

参数	类型	默认	必需	描述
`enable_gt_routing`	布尔	`false`	否	除了 PC 路由外启用 GT 路由

示例：

```
config :omniss7,
  enable_gt_routing: true
```

全局标题 **NAT** 参数

全局标题网络地址转换允许根据呼叫方前缀使用不同的响应 GT。有关详细说明和示例，请参见 全局标题 NAT 指南。

参数	类型	默认	必需	描述
`gt_nat_enabled`	布尔	`false`	否	启用/禁用 GT NAT 功能
`gt_nat_rules`	映射列表	`[]`	是（如果启用）	前缀到 GT 的映射列表

规则格式：`gt_nat_rules` 中的每个规则必须是一个映射，包含：

- `calling_prefix`：字符串前缀，用于与呼叫 GT 匹配
- `response_gt`：用于响应的全局标题

示例：

```
config :omniss7,
  gt_nat_enabled: true,
  gt_nat_rules: [
    # 当从以 "8772" 开头的 GT 被叫时，响应 "55512341112"
    %{calling_prefix: "8772", response_gt: "55512341112"},
    # 当从以 "8773" 开头的 GT 被叫时，响应 "55512341111"
    %{calling_prefix: "8773", response_gt: "55512341111"},
    # 默认回退（空前缀匹配所有）
    %{calling_prefix: "", response_gt: "55512311555"}
  ]
```

另见：有关详细用法和示例，请参见 GT NAT 指南。

M3UA 连接参数

MAP 客户端模式的 M3UA 连接配置。有关详细用法和示例，请参见 MAP 客户端指南。

□□	□□	□□	□□
`map_client_m3ua.mode`	□□□□	-	□	□□□□□ `"ASP"` □ `"SGP"`
`map_client_m3ua.callback`	□□	-	□	□□□□/□□□ `{MapClient, :handle_payload, []}`
`map_client_m3ua.process_name`	□□	-	□	□□□□□□□□
`map_client_m3ua.local_ip`	□□□□□□□	-	□	□□□□□□□ IP □□□□□□□ `{10, 0, 0, 1}` □□□□ □□ SCTP □□□□□ `[{10, 0, 0, 1}, {10, 0, 0, 2}]`
`map_client_m3ua.local_port`	□□□	`2905`	□	□□ SCTP □□
`map_client_m3ua.remote_ip`	□□□□□□□	-	□	□□ STP/SGW IP □□□□□□ `{10, 0, 0, 10}` □□□□□ `[{10, 0, 0, 10}, {10, 0, 0, 11}]`
`map_client_m3ua.remote_port`	□□□□	`2905`	□	□□ SCTP □□
`map_client_m3ua.routing_context`	□□□□□	-	□	M3UA □□□□□□ ID

□□□□□□ **IP**□:

```
config :omniss7,
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :hlr_client_asp,
    local_ip: {10, 179, 4, 11},
    local_port: 2905,
    remote_ip: {10, 179, 4, 10},
    remote_port: 2905,
    routing_context: 1
  }
```

示例（**SCTP** 多宿主）：

```
config :omniss7,
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :hlr_client_asp,
    # 多个本地 IP 用于冗余
    local_ip: [{10, 179, 4, 11}, {10, 179, 4, 12}],
    local_port: 2905,
    # 多个远程 IP 用于 STP 冗余
    remote_ip: [{10, 179, 4, 10}, {10, 179, 4, 20}],
    remote_port: 2905,
    routing_context: 1
  }
```

> 注意：有关 SCTP 多宿主配置和好处的详细信息，请参见 常见指南中的 SCTP 多宿主。

HTTP 服务器参数

REST API HTTP 服务器的配置。

参数	类型	默认	必需	描述
`start_http_server`	布尔	`true`	否	启用/禁用 HTTP 服务器（端口 8080）

硬编码值（不可配置）：

- **IP**: 0.0.0.0（所有接口）
- 端口: 8080
- 传输: Plug.Cowboy

示例：

```
config :omniss7,
  start_http_server: true  # 置为 false 以禁用
```

API 端点：

- REST API: `http://[server-ip]:8080/api/*`
- Swagger UI: `http://[server-ip]:8080/swagger`
- Prometheus 指标: `http://[server-ip]:8080/metrics`

有关详细信息，请参见 API 指南。

数据库参数

Mnesia 数据库持久性的配置。

参数	类型	默认	必需	描述
`mnesia_storage_type`	原子	`:disc_copies`	否	Mnesia 存储类型: `:disc_copies` 或 `:ram_copies`

示例：

```
config :omniss7,
  mnesia_storage_type: :disc_copies  # 生产
  # mnesia_storage_type: :ram_copies  # 仅用于测试
```

存储类型：

- `:disc_copies` - 持久磁盘存储（重启后仍然存在） - 推荐用于生产
- `:ram_copies` - 仅在内存中（重启时丢失） - 仅用于测试

Mnesia 表：

- `m3ua_peer` - M3UA 对等连接
- `m3ua_route` - 点代码路由
- `m3ua_gt_route` - 全局标题路由

位置：`Mnesia.{node_name}/` 目录

硬编码值

以下值在源代码中硬编码，无法通过配置更改。

超时

值	影响	解决方法
MAP 请求超时：**10** 秒	所有 MAP 操作在 10 秒后超时	修改源代码
ISD 超时：**10** 秒	每个 ISD 消息在 10 秒后超时	修改源代码

HTTP 服务器

值	影响	解决方法
HTTP IP: **0.0.0.0**	服务器在所有接口上监听	修改源代码
HTTP 端口: **8080**	REST API 在端口 8080 上运行	修改源代码

SSL 验证

值	影响	解决方法
HLR API SSL: 禁用	SSL 验证始终禁用	修改源代码
SMSc API SSL: 禁用	SSL 验证始终禁用	修改源代码

注册间隔

值	影响	解决方法
前端注���: **5** 分钟	SMSc 每 5 分钟与后端注册一次	修改源代码

Web UI 自动刷新

页面	间隔
路由管理	5 秒
活跃用户	2 秒

□□□□

□□ HLR □□

```
config :omniss7,
  map_client_enabled: true,
  hlr_mode_enabled: true,
  smsc_mode_enabled: false,

  hlr_api_base_url: "https://10.180.2.140:8443",
  hlr_service_center_gt_address: "55512341111",
  smsc_service_center_gt_address: "55512341112",

  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :hlr_client_asp,
    local_ip: {10, 179, 4, 11},
    local_port: 2905,
    remote_ip: {10, 179, 4, 10},
    remote_port: 2905,
    routing_context: 1
  }
```

□□ SMSc □□

```
config :omniss7,
  map_client_enabled: true,
  hlr_mode_enabled: false,
  smsc_mode_enabled: true,

  smsc_api_base_url: "https://10.179.3.219:8443",
  smsc_name: "ipsmgw",
  smsc_service_center_gt_address: "55512341112",

  auto_flush_enabled: true,
  auto_flush_interval: 10_000,
  auto_flush_dest_smsc: "ipsmgw",
  auto_flush_tps: 10,

  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :stp_client_asp,
    local_ip: {10, 179, 4, 12},
    local_port: 2905,
    remote_ip: {10, 179, 4, 10},
    remote_port: 2905,
    routing_context: 1
  }
```

使用独立服务器的 **STP**

```
config :omniss7,
  map_client_enabled: true,
  hlr_mode_enabled: false,
  smsc_mode_enabled: false,

  enable_gt_routing: true,
  mnesia_storage_type: :disc_copies,

  m3ua_stp: %{
    enabled: true,
    local_ip: {10, 179, 4, 10},
    local_port: 2905,
    point_code: 100
  },

  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :stp_client_asp,
    local_ip: {10, 179, 4, 10},
    local_port: 2906,
    remote_ip: {10, 179, 4, 11},
    remote_port: 2905,
    routing_context: 1
  }
```

摘要

总配置参数: 32

按类别:

- 操作模式: 3 个参数
- HLR 模式: 13 个参数
- SMSc 模式: 7 个参数

- STP 模式: 5 个参数
- M3UA 连接: 8 个参数
- HTTP 服务器: 1 个参数
- 数据库: 1 个参数

必需参数（必须设置）：

- `hlr_api_base_url`（HLR 模式）
- `hlr_service_center_gt_address`（HLR 模式）
- `smsc_api_base_url`（SMSc 模式）
- `smsc_service_center_gt_address`（SMSc/HLR 模式）
- 所有 `map_client_m3ua.*` 参数
- `m3ua_stp.point_code`（如果启用 STP）

相关文档

- **HLR □南** - HLR 特定配置
- **SMSc □南** - SMSc 特定配置
- **STP □南** - STP 路由配置
- **API □南** - REST API 参考
- **Web UI □南** - Web 界面文档

全球标题 NAT □南

概述

全球标题网络地址转换（GT NAT）是一项功能，允许 OmniSS7 根据呼叫方的 GT 前缀、被叫方的 GT 前缀或两者的组合响应不同的全球标题地址。当需要确保响应使用正确的 GT 时，这一点至关重要，尤其是在与多个全球标题操作时，确保根据哪个网络或对等方在呼叫和/或他们呼叫的 GT 来使用正确的 GT。

新功能（增强的 GT NAT）

GT NAT 功能已增强，具备强大的新能力：

新特性

1. 被叫方前缀匹配：规则现在可以匹配 `called_prefix`，除了 `calling_prefix` 之外
2. 组合匹配：规则可以同时匹配呼叫和被叫前缀
3. 基于权重的优先级：规则现在使用 `weight` 字段（较低 = 较高优先级），而不仅仅是前缀长度
4. 灵活匹配：现在可以创建具有以下特性的规则：
    - 仅呼叫前缀
    - 仅被叫前缀
    - 同时具有呼叫和被叫前缀
    - 两者都没有（通配符/回退规则）

新规则格式

必需字段：

- `weight`：整数优先级（较低 = 较高优先级）
- `response_gt`：要响应的 GT

可选字段（建议至少□定一个以进行特定匹配）：

- `calling_prefix`：匹配呼叫方 GT 前缀
- `called_prefix`：匹配被叫方 GT 前缀

示例：

```
gt_nat_rules: [
  # 同时具有两个前缀的特定规则 - 最高优先级
  %{calling_prefix: "8772", called_prefix: "555", weight: 1,
response_gt: "111111"},

  # 特定规则 - 中等优先级
  %{calling_prefix: "8772", weight: 10, response_gt: "222222"},
  %{called_prefix: "555", weight: 10, response_gt: "333333"},

  # 通配符回退 - 最低优先级
  %{weight: 100, response_gt: "999999"}
]
```

用例

多网络操作

当您有多个对等网络，每个网络都期望从特定 GT 接收响应时：

- 网络 **A** 呼叫您的 GT `111111` 并期望从 `111111` 接收响应
- 网络 **B** 呼叫您的 GT `222222` 并期望从 `222222` 接收响应

没有 GT NAT，您需要单独的实例或复杂的路由。使用 GT NAT，单个 OmniSS7 实例可以智能地处理此情况。

漫游场景

当作为 HLR 或 SMSc 运营时，具有漫游协议：

- 本地网络用户使用 GT `555000`
- 漫游合作伙伴 **1** 使用 GT `555001`
- 漫游合作伙伴 **2** 使用 GT `555002`

GT NAT 确保每个合作伙伴从他们配置的正确 GT 接收响应。

测试和迁移

在网络迁移或测试期间：

- 逐步将流量从旧 GT 迁移到新 GT
- 在过渡期间维护两个 GT
- 根据呼叫者使用的 GT 路由响应

工作原理

地址转换流程

1. 传入请求：OmniSS7 接收到带有以下内容的 SCCP 消息：
    - 被叫方 GT：`55512341112`（您的 GT）
    - 呼叫方 GT：`877234567`（他们的 GT）
2. **GT NAT** 查找：系统检查呼叫 GT `877234567` 是否与配置的前缀规则匹配
3. 前缀匹配：找到最长的匹配前缀（例如，`8772` 匹配 `877234567`）
4. 响应 **GT** 选择：使用匹配规则中的 `response_gt`（例如，`55512341112`）
5. 发送响应：SCCP 响应使用：
    - 被叫方 GT：`877234567`（反转 - 他们的 GT）
    - 呼叫方 GT：`55512341112`（NAT'd GT）

受影响的响应类型

GT NAT 适用于 SS7 栈的多个层次：

SCCP 层（所有响应）

- 所有响应消息中的 SCCP 被叫/呼叫 GT 地址
- ISD（InsertSubscriberData）确认
- UpdateLocation 响应

- 错误响应

MAP 层（操作特定）

- **SRI-for-SM** 响应：`networkNode-Number`（SMSc GT 地址）
- **UpdateLocation**：响应中的 `hlr-Number`
- **InsertSubscriberData**：ISD 消息中的 HLR GT

配置

基本配置

添加到 `config/runtime.exs`：

```
config :omniss7,
  # 启用 GT NAT
  gt_nat_enabled: true,

  # 定义 GT NAT 规则
  gt_nat_rules: [
    # 规则 1：来自前缀 "8772" 的呼叫从 "55512341112" 获取响应
    %{calling_prefix: "8772", response_gt: "55512341112"},

    # 规则 2：来自前缀 "8773" 的呼叫从 "55512341111" 获取响应
    %{calling_prefix: "8773", response_gt: "55512341111"},

    # 默认规则（空前缀匹配所有内容）
    %{calling_prefix: "", response_gt: "55512311555"}
  ]
```

配置参数

有关完整的配置参考，请参见 配置参考中的全球标题 NAT 参数。

参数	类型	必需	描述
`gt_nat_enabled`	布尔	是	启用/禁用 GT NAT 功能
`gt_nat_rules`	映射列表	是（如果启用）	前缀匹配规则列表

规则格式

每个规则都是一个具有以下键的映射：

```
%{
  calling_prefix: "8772",       # （可选）要与呼叫 GT 匹配的前缀
  called_prefix: "555",         # （可选）要与被叫 GT 匹配的前缀
  weight: 10,                   # （必需）优先级值（较低 = 较高优先级）
  response_gt: "55512341112"    # （必需）在响应中使用的 GT
}
```

规则字段：

- **`calling_prefix`**（可选）：要与传入呼叫 GT 匹配的字符串前缀
  - 匹配通过 `String.starts_with?/2` 完成
  - 空字符串 `""` 或 `nil` 作为通配符（匹配任何呼叫 GT）
  - 可以省略以匹配任何呼叫 GT
- **`called_prefix`**（可选）：要与传入被叫 GT 匹配的字符串前缀
  - 匹配通过 `String.starts_with?/2` 完成
  - 空字符串 `""` 或 `nil` 作为通配符（匹配任何被叫 GT）
  - 可以省略以匹配任何被叫 GT
- **`weight`**（必需）：整数优先级值
  - 较低的权重 = 较高的优先级（优先处理）
  - 必须大于或等于 0
  - 用作匹配规则的主要排序标准

- **`response_gt`**（必需）：在响应中使用的全球标题地址
    - 必须是有效的 E.164 数字字符串
    - 应与您的配置 GT 之一匹配

至少应□定 **`calling_prefix`** 或 **`called_prefix`** 中的一个以进行特定路由。两者都可以省略以用于通配符/回退规则。

规则匹配逻辑

规则首先按 权重（升序）进行评估，然后按 组合前缀特异性：

匹配算法：

1. 筛选所有指定前缀匹配的规则
    - 如果设置了 `calling_prefix`，则必须匹配呼叫 GT
    - 如果设置了 `called_prefix`，则必须匹配被叫 GT
    - 如果两者都设置，则两者都必须匹配
    - 如果两者都未设置，则规则作为通配符
2. 按以下方式对匹配规则进行排序：
    - 主要：权重（升序 - 较低的值优先）
    - 次要：组合前缀长度（降序 - 较长 = 更特定）
3. □回第一个匹配的规则

示例：

```
# 示例规则
gt_nat_rules: [
  # 权重 1: 最高优先级 - 匹配两个前缀
  %{calling_prefix: "8772", called_prefix: "555", weight: 1, response_gt: "111111"},

  # 权重 10: 中等优先级 - 特定规则
  %{calling_prefix: "8772", weight: 10, response_gt: "222222"},  # 仅呼叫
  %{called_prefix: "555", weight: 10, response_gt: "333333"},    # 仅被叫

  # 权重 100: 最低优先级 - 通配符回退
  %{weight: 100, response_gt: "444444"}  # 匹配所有
]

# 匹配示例:
# 呼叫: "877234567", 被叫: "555123" -> "111111" (权重 1, 两个匹配)
# 呼叫: "877234567", 被叫: "999999" -> "222222" (权重 10, 仅呼叫)
# 呼叫: "999999999", 被叫: "555123" -> "333333" (权重 10, 仅被叫)
# 呼叫: "999999999", 被叫: "888888" -> "444444" (权重 100, 通配符)
```

示例

示例 1：两个网络合作伙伴

场景：您运营一个 SMSc，拥有两个网络合作伙伴。每个合作伙伴期望从不同的 GT 接收响应。

```
config :omniss7,
  gt_nat_enabled: true,

  # 默认 SMSc GT（在禁用 GT NAT 或没有规则匹配时使用）
  smsc_service_center_gt_address: "5551000",

  # 合作伙伴的 GT NAT 规则
  gt_nat_rules: [
    # 合作伙伴 A（前缀 4412）期望从 GT 5551001 获取响应
    %{calling_prefix: "4412", weight: 10, response_gt: "5551001"},

    # 合作伙伴 B（前缀 4413）期望从 GT 5551002 获取响应
    %{calling_prefix: "4413", weight: 10, response_gt: "5551002"},

    # 默认：使用标准 SMSc GT（通配符回退）
    %{weight: 100, response_gt: "5551000"}
  ]
```

流量流程：

```
来自 44121234567 的传入 SRI-for-SM：
  被叫 GT：5551001（您使用的 GT，合作伙伴 A 使用）
  呼叫 GT：44121234567（合作伙伴 A 的 GT）

GT NAT 查找：
  "44121234567" 匹配前缀 "4412"
  选择的 response_gt： "5551001"

响应 SRI-for-SM 到 44121234567：
  被叫 GT：44121234567（反转）
  呼叫 GT：5551001（NAT'd）
  networkNode-Number：5551001（在 MAP 响应中）
```

示例 2：具有区域 GT 的 HLR

场景：国家 HLR，每个区域有不同的 GT。

```
config :omniss7,
  gt_nat_enabled: true,
  hlr_service_center_gt_address: "555000",  # 默认 HLR GT

  gt_nat_rules: [
    # 北方区域 VLR（前缀 5551）
    %{calling_prefix: "5551", weight: 10, response_gt: "555100"},

    # 南方区域 VLR（前缀 5552）
    %{calling_prefix: "5552", weight: 10, response_gt: "555200"},

    # 西方区域 VLR（前缀 5553）
    %{calling_prefix: "5553", weight: 10, response_gt: "555300"},

    # 其他区域的默认（通配符）
    %{weight: 100, response_gt: "555000"}
  ]
```

示例 3：迁移场景

场景：逐步从旧 GT 迁移到新 GT。

```
config :omniss7,
  gt_nat_enabled: true,
  hlr_service_center_gt_address: "123456789",  # 旧 GT（默认）

  gt_nat_rules: [
    # 已迁移的网络（已更新其配置）
    %{calling_prefix: "555", weight: 10, response_gt:
"987654321"},  # 新 GT
    %{calling_prefix: "666", weight: 10, response_gt:
"987654321"},  # 新 GT

    # 其他人仍然使用旧 GT（通配符）
    %{weight: 100, response_gt: "123456789"}  # 旧 GT
  ]
```

示例 4：被叫方前缀匹配（新）

场景：您有多个GT用于不同服务，并希望根据呼叫的GT响应正确的GT。

```
config :omniss7,
  gt_nat_enabled: true,

  gt_nat_rules: [
    # 当他们呼叫您的 SMS GT（5551xxx）时，响应该 GT
    %{called_prefix: "5551", weight: 10, response_gt: "555100"},

    # 当他们呼叫您的语音 GT（5552xxx）时，响应该 GT
    %{called_prefix: "5552", weight: 10, response_gt: "555200"},

    # 当他们呼叫您的数据 GT（5553xxx）时，响应该 GT
    %{called_prefix: "5553", weight: 10, response_gt: "555300"},

    # 默认回退
    %{weight: 100, response_gt: "555000"}
  ]
```

流量流程：

```
传入请求到被叫 GT：555100（您的 SMS GT）
呼叫 GT：441234567（任何呼叫者）

GT NAT 查找：
  被叫 GT "555100" 匹配前缀 "5551"
  选择的 response_gt： "555100"

响应使用呼叫 GT：555100（匹配他们所呼叫的）
```

示例 5：组合呼叫 + 被叫前缀匹配（高级）

场景：不同的合作伙伴呼叫不同的GT，您希望进行精细控制。

```
config :omniss7,
  gt_nat_enabled: true,

  gt_nat_rules: [
    # □□□□□ A □□□□□ SMS GT - □□□□□□□□□□ 1□
    %{calling_prefix: "4412", called_prefix: "5551", weight: 1,
response_gt: "555101"},

    # □□□□□ B □□□□□ SMS GT - □□□□□□□□□□ 1□
    %{calling_prefix: "4413", called_prefix: "5551", weight: 1,
response_gt: "555102"},

    # □□□□□□□□□ SMS GT - ���□□□□□□□□□ 10□
    %{called_prefix: "5551", weight: 10, response_gt: "555100"},

    # □□□□□ A □□□□□ GT - □□□□□□□□□□ 10□
    %{calling_prefix: "4412", weight: 10, response_gt: "555200"},

    # □□□□□ - □□□□□□□□□ 100□
    %{weight: 100, response_gt: "555000"}
  ]
```

□□□□□□

```
# □□□□□ A □□ SMS GT
□□□"441234567"□□□□□"555100"
→ □□□□□ 1 □□□□□□□□□→ "555101"

# □□□□□ A □□□□□ GT
□□□"441234567"□□□□□"555200"
→ □□□□□ 10 □□□□□□□□→ "555200"

# □□□□□□□□□ SMS GT
□□□"999999999"□□□□□"555100"
→ □□□□□ 10 □□□□□□□□→ "555100"

# □□□□□□□□□□□ GT
□□□"999999999"□□□□□"555200"
→ □□□□□ 100 □□□ → "555000"
```

操作模式

GT NAT 在所有 OmniSS7 操作模式下工作:

HLR 模式

GT NAT 影响:

- UpdateLocation 响应（响应中的 HLR GT）
- InsertSubscriberData 消息（作为呼叫方的 HLR GT）
- SendAuthenticationInfo 响应
- 取消位置响应

有关 HLR 操作的更多信息，请参见 HLR 配置指南。

配置:

```
config :omniss7,
  hlr_mode_enabled: true,
  hlr_service_center_gt_address: "5551234567",  # 默认 HLR GT

  gt_nat_enabled: true,
  gt_nat_rules: [
    %{calling_prefix: "331", weight: 10, response_gt: "5551234568"},  # 法国
    %{calling_prefix: "44", weight: 10, response_gt: "5551234569"},   # 英国
    %{weight: 100, response_gt: "5551234567"}  # 默认通配符
  ]
```

SMSc 模式

GT NAT 影响:

- SRI-for-SM 响应（`networkNode-Number` 字段） - 参见 SRI-for-SM 详细信息
- MT-ForwardSM 确认

有关 SMSc 操作的更多信息，请参见 SMSc 配置指南。

配置:

```
config :omniss7,
  smsc_mode_enabled: true,
  smsc_service_center_gt_address: "5559999",  # 默认 SMSc GT

  gt_nat_enabled: true,
  gt_nat_rules: [
    %{calling_prefix: "1", weight: 10, response_gt: "5559991"},
# 北美
    %{calling_prefix: "44", weight: 10, response_gt: "5559992"},
# 英国
    %{calling_prefix: "86", weight: 10, response_gt: "5559993"},
# 中国
    %{weight: 100, response_gt: "5559999"}  # 默认通配符
  ]
```

CAMEL 网关模式

GT NAT 影响:

- 所有 SCCP 层响应（gsmSCF GT 作为呼叫方）
- CAMEL/CAP 操作响应（InitialDP、EventReportBCSM 等）
- RequestReportBCSMEvent 确认
- ApplyCharging ���应
- Continue 响应

配置:

```
config :omniss7,
  camelgw_mode_enabled: true,
  camel_gsmscf_gt_address: "55512341112",  # □□ gsmSCF GT

  gt_nat_enabled: true,
  gt_nat_rules: [
    %{calling_prefix: "555", weight: 10, response_gt:
"55512341111"},  # □□ A
    %{calling_prefix: "666", weight: 10, response_gt:
"55512311555"},  # □□ B
    %{weight: 100, response_gt: "55512341112"}  # □□□□□
  ]
```

□□□ □□□□□□□□□ gsmSCF□□□□□□□□□□□□□□□□□□□ gsmSSF □□□□□□□□□ gsmSCF GT □□□ □□GT NAT □□□□□□□ gsmSSF □□□□□□□□□ GT□

□□□□□□□

□□ GT NAT □□□□

GT NAT □□□□□□□□□□□□□□□□□

```
# □□□□□□□□□□□□
[info] GT NAT [SRI-for-SM □□]: □□ GT 877234567 -> □□ GT
55512341112
[info] GT NAT [UpdateLocation ISD]: □□ GT 331234567 -> □□ GT
55512341111
[info] GT NAT [MAP BEGIN □□]: □□ GT 441234567 -> □□ GT 55512311555
```

□□□□□□□□□ NAT □□□□□□□

- `"SRI-for-SM □□"` - □ SRI-for-SM □□□□□
- `"UpdateLocation ISD"` - □ InsertSubscriberData □□□
- `"UpdateLocation END"` - □ UpdateLocation END □□□
- `"MAP BEGIN □□"` - ��□ MAP BEGIN □□
- `"ISD ACK"` - ISD □□

- `"HLR 错误响应"` - HLR 的错误响应
- `"CAMEL 响应"` - CAMEL/CAP 操作响应（gsmSCF）

验证

系统在启动时验证 GT NAT 配置：

```
# 检查 GT NAT 配置
iex> GtNat.validate_config()
{:ok, [
  %{calling_prefix: "8772", weight: 10, response_gt:
"55512341112"},
  %{calling_prefix: "8773", weight: 10, response_gt:
"55512341111"}
]}

# 检查是否启用
iex> GtNat.enabled?()
true

# 获取所有规则
iex> GtNat.get_rules()
[
  %{calling_prefix: "8772", weight: 10, response_gt:
"55512341112"},
  %{calling_prefix: "8773", weight: 10, response_gt:
"55512341111"}
]
```

测试 GT NAT

以编程方式测试 GT NAT 逻辑：

```
# 仅使用呼叫 GT 测试转换（called_gt 为 nil）
iex> GtNat.translate_response_gt("877234567", nil, "default_gt")
"55512341112"

# 使用呼叫和被叫 GT 测试转换
iex> GtNat.translate_response_gt("877234567", "555123",
"default_gt")
"55512341112"

# 测试日志记录（nil 被叫 GT）
iex> GtNat.translate_response_gt_with_logging("877234567", nil,
"default_gt", "test")
# 日志: GT NAT [test]: 呼叫 GT 877234567 -> 响应 GT 55512341112
"55512341112"

# 测试日志记录（两个 GT）
iex> GtNat.translate_response_gt_with_logging("877234567",
"555123", "default_gt", "test")
# 日志: GT NAT [test]: 呼叫 GT 877234567, 被叫 GT 555123 -> 响应 GT
55512341112
"55512341112"

# 测试无匹配（返回默认）
iex> GtNat.translate_response_gt("999999999", "888888",
"default_gt")
"default_gt"
```

故障排除

问题: **GT NAT** 无法工作

检查 **1**: 是否启用?

```
iex> Application.get_env(:omniss7, :gt_nat_enabled)
true  # 应为 true
```

检查 **2**: 规则是否配置?

```
iex> Application.get_env(:omniss7, :gt_nat_rules)
[%{calling_prefix: "8772", response_gt: "55512341112"}, ...]  # 应返回列表
```

检查 **3**：检查日志 搜索日志中的 "GT NAT" 以查看是否发生了转换。

问题：响应中的 **GT** 错误

症状：响应使用意外的 GT 地址

原因：规则前缀匹配可能过于宽泛或默认规则捕获了流量

解决方案：检查规则权重和前缀：

```
# 不好：低权重的通配符（捕获所有内容）
gt_nat_rules: [
  %{weight: 1, response_gt: "111111"},               # 这会首先匹配所有内容！
  %{calling_prefix: "8772", weight: 10, response_gt: "222222"}  # 从未到达
]

# 好：具有较低权重的特定规则，较高权重的通配符
gt_nat_rules: [
  %{calling_prefix: "8772", weight: 10, response_gt: "222222"},  # 特定，低权重
  %{weight: 100, response_gt: "111111"}  # 通配符，高权重（回退）
]
```

问题：**GT NAT** 未应用于特定消息类型

症状：某些响应使用 NAT'd GT，其他则不使用

当前覆盖范围：

- □ SCCP 呼叫 GT（所有响应）
- □ SRI-for-SM 响应（networkNode-Number）
- □ UpdateLocation ISD 消息（HLR GT）
- □ UpdateLocation END 响应

- □ ISD 确认
- □ MAP BEGIN 响应

如果特定消息类型未使用 GT NAT，可能尚未实现。检查源代码或联系支持。

性能考虑

查找性能

GT NAT 使用简单的前缀匹配，复杂度为 O(n)，其中 n 是规则的数量。

性能提示：

- 将规则数量保持在 100 以下以获得最佳性能
- 使用特定前缀以减少规则数量
- 默认规则（空前缀）应放在最后

基准（典型系统）：

- 10 条规则：每次查找 < 1μs
- 50 条规则：每次查找 < 5μs
- 100 条规则：每次查找 < 10μs

内存使用

每条规则大约需要 100 字节的内存：

- 10 条规则 ≈ 1 KB
- 100 条规则 ≈ 10 KB

最佳实践

1. 始终包含通配符回退规则

```
gt_nat_rules: [
  %{calling_prefix: "8772", weight: 10, response_gt: "111111"},
  %{calling_prefix: "8773", weight: 10, response_gt: "222222"},
  %{weight: 100, response_gt: "default_gt"}  # 始终有一个高权重的通配符
]
```

2. 使用有意义的前缀和权重

```
# 好：清晰、特定的前缀和适当的权重
%{calling_prefix: "331", weight: 10, response_gt: "..."}  # 法国
%{calling_prefix: "44", weight: 10, response_gt: "..."}   # 英国

# 不好：过于宽泛的前缀或混淆的权重
%{calling_prefix: "3", weight: 5, response_gt: "..."}     # 太多国家
%{calling_prefix: "331", weight: 100, response_gt: "..."}  # 权重应为较低以适用于特定规则
```

3. 记录您的规则

```
gt_nat_rules: [
  # 合作伙伴 XYZ - 英国网络（GT 范围：4412xxxxxxx）
  # 权重 10：标准合作伙伴优先级
  %{calling_prefix: "4412", weight: 10, response_gt: "5551001"},

  # 合作伙伴 ABC - 法国网络（GT 范围：33123xxxxxx）
  # 权重 10：标准合作伙伴优先级
  %{calling_prefix: "33123", weight: 10, response_gt: "5551002"}
]
```

4. 部署前测试

```
# 在部署前 iex 中测试
iex> GtNat.translate_response_gt("44121234567", nil, "default")
"5551001"  # 预期结果

# 使用被叫 GT 测试
iex> GtNat.translate_response_gt("44121234567", "555123",
"default")
"5551001"  # 预期结果
```

5. 监控日志

启用 INFO 级别日志记录，以查看生产环境中的所有 GT NAT 转换。

与其他功能的集成

STP 模式

GT NAT 独立于 STP 路由工作。STP 根据点代码和目标 GT 进行路由，而 GT NAT 处理响应地址。

有关 STP 路由的更多信息，请参见 STP 配置指南。

CAMEL 集成

GT NAT 与 CAMEL/CAP 操作完全集成：

SCCP 层：

- 所有 CAMEL 响应中的呼叫方 GT
- 根据传入的 gsmSSF GT 自动应用

配置：

- `camel_gsmscf_gt_address` - 默认 gsmSCF GT（可选）
- 如果未配置，则使用来自传入请求的被叫方 GT
- GT NAT 规则根据呼叫方前缀覆盖默认值

示例：

```
# 当 gsmSSF 555123456 呼叫您的 gsmSCF
# 传入：被叫=55512341112，呼叫=555123456
# GT NAT 查找："555" -> response_gt="55512341111"
# 响应：被叫=555123456，呼叫=55512341111
```

负载均衡

GT NAT 可以与 M3UA 负载均衡结合使用，以实现高级流量管理。

迁移□南

在现有系统上启用 GT NAT

1. 准备配置

```
# 添加到 runtime.exs（最初保持禁用）
config :omniss7,
  gt_nat_enabled: false,  # 初始禁用
  gt_nat_rules: [
    # 您的规则在这里，带有权重
    %{calling_prefix: "877", weight: 10, response_gt:
"111111"},
    %{weight: 100, response_gt: "999999"}  # 通配符回退
  ]
```

2. 测试配置

```
# 验证配置是否编译
mix compile

# 在 iex 中测试
iex -S mix
iex> GtNat.validate_config()
```

3. 在暂存环境中启用

```
gt_nat_enabled: true  # 更改为 true
```

4. 监控日志

```
tail -f log/omniss7.log | grep "GT NAT"
```

5. 部署到生产

    - 在维护窗口期间进行部署
    - 密切监控前 24 小时
    - 准备回滚计划（设置 `gt_nat_enabled: false`）

支持

如有问题或疑问：

- 检查日志中的 "GT NAT" 消息
- 使用 `GtNat.validate_config()` 验证配置
- 查看本指南的故障排除部分
- 联系 OmniSS7 支持，提供日志摘录

另见

- HLR 指南 - HLR 模式配置
- SMSC 指南 - SMSc 模式配置
- STP 指南 - STP 路由配置
- 配置参考 - 完整配置参考

HLR 配置□南

← □回主文档

本指南提供了将 OmniSS7 作为 家庭位置寄存器 **(HLR/HSS)** 与 **OmniHSS** 作为后端用户数据库的配置。

OmniHSS 集成

OmniSS7 HLR 模式作为 **SS7** 信令前端，与 **OmniHSS**（一个功能齐全的家庭用户服务器 HSS 后端）进行接口。该架构分离了关注点：

- **OmniSS7 (HLR** 前端**)**：处理所有 SS7/MAP 协议信令、SCCP 路由和网络通信
- **OmniHSS (HSS** 后端**)**：管理用户数据、身份验证、配置和高级功能

为什么选择 OmniHSS？

OmniHSS 提供了运营商级的用户管理，具有以下功能：

- 多 **IMSI** 支持：每个用户可以与一个 MSISDN 关联多个 IMSI，以便于国际漫游、网络切换和 eSIM 配置
- 灵活的身份验证：支持 Milenage（3G/4G/5G）和 COMP128（2G）身份验证算法
- 电路和数据会话跟踪：独立跟踪 CS（电路交换）和 PS（分组交换）网络注册
- 高级配置：可定制的服务配置、附加服务和 CAMEL 订阅数据
- **API** 优先设计：RESTful HTTP API 用于与计费、CRM 和配置系统的集成
- 实时更新：位置跟踪、会话管理和身份验证向量生成

所有用户数据、身份验证凭据和服务配置都存储在 OmniHSS 中。OmniSS7 通过 HTTPS API 调用查询 OmniHSS，以响应 MAP 操作，如 UpdateLocation、SendAuthenticationInfo 和 SendRoutingInfo。

重要：OmniSS7 HLR 模式仅为 信令前端。所有用户管理逻辑、身份验证算法、配置规则和数据库操作均由 OmniHSS 处理。本指南涵盖 OmniSS7 中的 SS7/MAP 协议配置。有关用户配置、身份验证配置、服务配置和管理操作的信息，请 参阅 **OmniHSS** 文档。

多 IMSI 支持

OmniHSS 原生支持多 **IMSI** 配置，允许单个用户（通过 MSISDN 识别）拥有多个 IMSI。这使得：

- 国际漫游配置：不同区域的不同 IMSI 以降低漫游费用
- **eSIM** 多配置：在单个支持 eSIM 的设备上使用多个网络配置
- 网络切换：在不更改 MSISDN 的情况下无缝切换网络
- 双 **SIM** 协调：跨多个物理或虚拟 SIM 进行协调
- 测试与开发：多个测试 IMSI 指向同一用户

工作原理：

- 每个 IMSI 具有自己的身份验证凭据（Ki、OPc、算法）
- 每个 IMSI 可以有独立的电路和数据会话注册
- 用户服务和配置可以按 IMSI 共享或自定义
- OmniSS7 按 IMSI 查询 OmniHSS，OmniHSS 回适当的用户数据
- 计费系统可以按 IMSI 跟踪使用情况，同时将所有 IMSI 关联到单一账户

示例多 **IMSI** 场景：

```
用户 MSISDN: +1-555-123-4567
├─ IMSI 1: 310260123456789 (美国本地网络 - Milenage 身份验证)
├─ IMSI 2: 208011234567890 (法国漫游配置 - Milenage 身份验证)
└─ IMSI 3: 440201234567891 (英国漫游配置 - COMP128 身份验证)
```

所有三个 IMSI 可以独立用于网络注册，但它们都属于同一用户账户。OmniHSS 管理 IMSI 到用户的映射，并确保每个 IMSI 的适当身份验证和配置。

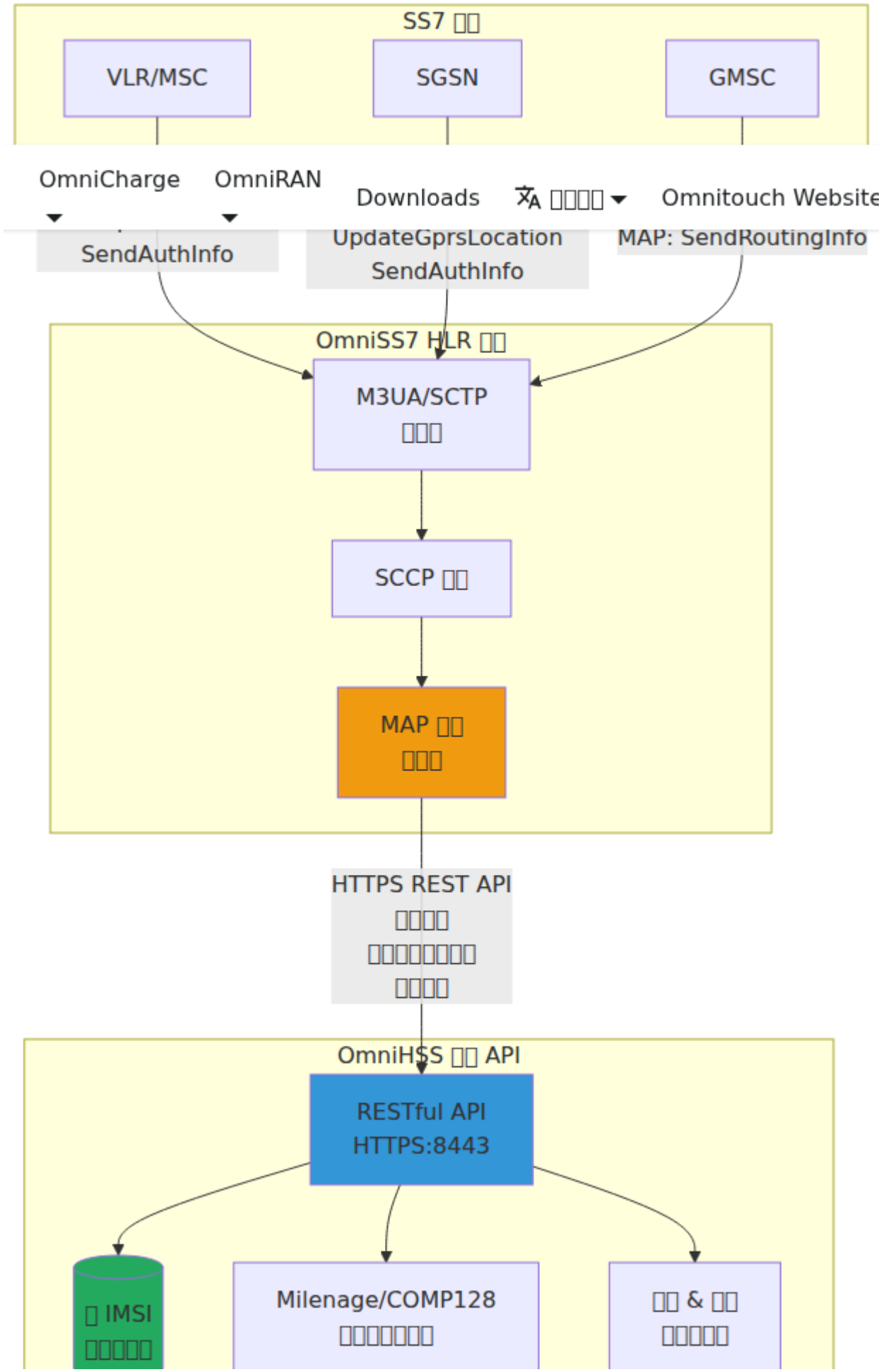
目录

什么是 HLR 模式?

HLR 模式使 OmniSS7 能够作为家庭位置寄存器运行:

- 用户管理: 存储和管理用户数据
- 身份验证: 生成网络访问的身份验证向量
- 位置跟踪: 处理来自 VLR 的位置更新
- 路由信息: 提供呼叫和 SMS 的路由信息

HLR □□


SS7 □□
VLR/MSC
SGSN
GMSC
OmniCharge
OmniRAN
Downloads
□□□□
Omnitouch Website
SendAuthInfo
UpdateGprsLocation
SendAuthInfo
MAP: SendRoutingInfo
OmniSS7 HLR □□
M3UA/SCTP
□□□
SCCP □□
MAP □□
□□□
HTTPS REST API
□□□□
□□□□□□□□□
□□□□
OmniHSS □□ API
RESTful API
HTTPS:8443
□ IMSI
□□□□□
Milenage/COMP128
□□□□□□□
□□ & □□
□□□□□

启用 **HLR** 模式

OmniSS7 可以在不同模式下运行。要将其用作 HLR，您需要在配置中启用 HLR 模式。

切换到 **HLR** 模式

OmniSS7 的 `config/runtime.exs` 包含三个预配置的操作模式。要启用 HLR 模式：

1. 打开 `config/runtime.exs`
2. 查找三个配置部分（第 53-174 行）：
   - 配置 1：STP 模式（第 53-85 行）
   - 配置 2：HLR 模式（第 87-123 行）
   - 配置 3：SMSc 模式（第 125-174 行）
3. 注释掉当前活动的配置（在每行前添加 `#`）
4. 取消注释 HLR 配置（从第 87-123 行移除 `#`）
5. 根据需要自定义配置参数
6. 重启应用程序：`iex -S mix`

HLR 模式配置

完整的 HLR 配置如下所示：

```
config :omniss7,
  # 模式标志 - 仅启用 HLR 功能
  map_client_enabled: true,
  hlr_mode_enabled: true,
  smsc_mode_enabled: false,

  # OmniHSS 后端 API 配置
  hlr_api_base_url: "https://10.180.2.140:8443",

  # HLR 服务中心 GT 地址，用于 SMS 操作
  hlr_service_center_gt_address: "1234567890",

  # MSISDN ↔ IMSI 映射配置
  # 参见: MSISDN ↔ IMSI 映射部分以获取详细信息
  hlr_imsi_plmn_prefix: "50557",
  hlr_msisdn_country_code: "61",
  hlr_msisdn_nsn_offset: 0,
  hlr_msisdn_nsn_length: 9,

  # InsertSubscriberData 配置
  # 网络访问模式: :packetAndCircuit、:packetOnly 或 :circuitOnly
  isd_network_access_mode: :packetAndCircuit,

  # 发送 ISD #2（附加服务数据）
  isd_send_ss_data: true,

  # 发送 ISD #3（呼叫限制数据）
  isd_send_call_barring: true,

  # CAMEL 配置（用于 SendRoutingInfo 响应）
  # CAMEL 服务启动的服务密钥
  camel_service_key: 11_110,

  # CAMEL 触发检测点
  # 选项: :termAttemptAuthorized、:tBusy、:tNoAnswer、:tAnswer
  camel_trigger_detection_point: :termAttemptAuthorized,

  # 家庭 VLR 前缀
  # 被视为“家庭”网络的 VLR 地址前缀列表
  # 如果用户的 VLR 以这些前缀之一开头，则使用标准 SRI 响应
  # 否则，用户正在漫游，我们需要发送 PRN 以获取 MSRN
  home_vlr_prefixes: ["123456"],
```

```
# M3UA 连接配置
# 作为 ASP 连接以接收 MAP 操作（UpdateLocation、SendAuthInfo 等）
map_client_m3ua: %{
  mode: "ASP",
  callback: {MapClient, :handle_payload, []},
  process_name: :hlr_client_asp,
  # 本地端点（HLR 系统）
  local_ip: {10, 179, 4, 11},
  local_port: 2905,
  # 远程 STP 端点
  remote_ip: {10, 179, 4, 10},
  remote_port: 2905,
  routing_context: 1
}
```

需要自定义的配置参数

有关所有配置参数的完整参考，请参见 配置参考。

□□	□□	□□□	
`hlr_api_base_url`	□□□	*□□*	**Om**
`hlr_service_center_gt_address`	□□□	*□□*	□□ U □□ H
`smsc_service_center_gt_address`	□□□	*□□*	□ SR SMS
`hlr_smsc_alert_gts`	□□	`[]`	□ Up □ al □ SM
`hlr_alert_location_expiry_seconds`	□□	`172800`	□ SM aler □□□□
`hlr_imsi_plmn_prefix`	□□□	`"50557"`	MSIS PLM ↔ IM
`hlr_msisdn_country_code`	□□□	`"61"`	□□ I □□□□ IMSI
`hlr_msisdn_nsn_offset`	□□	`0`	□ MS □□□□ IMSI
`hlr_msisdn_nsn_length`	□□	`9`	□□□□ MSIS

□□	□□	□□□	
`isd_network_access_mode`	□□	`:packetAndCircuit`	Inse□□□
`isd_send_ss_data`	□□	`true`	□□ IS
`isd_send_call_barring`	□□	`true`	□□ IS
`camel_service_key`	□□	`11_110`	Sen CAM
`camel_trigger_detection_point`	□□	`:termAttemptAuthorized`	CAM
`home_vlr_prefixes`	□□	`["5551231"]`	□□□' □□□
`local_ip`	□□	*□□*	□□ H
`local_port`	□□	`2905`	□□ S
`remote_ip`	□□	*□□*	□□ S
`remote_port`	□□	`2905`	□□ S
`routing_context`	□□	`1`	M3U

启用 **HLR** 模式时会发生什么

当 `hlr_mode_enabled: true` 时，Web UI 将显示：

- □ **SS7** 事件 - 事件日志
- □ **SS7** 客户端 - MAP 操作测试
- □ **M3UA** - 连接状态
- □ **HLR** 链接 - HLR API 状态 + 用户管理 ← *□□□ HLR*
- □ □□ - 系统监控
- □ □□ - 配置查看器

路由、路由测试和 **SMSc** 链接 选项卡将被隐藏。

重要说明

- 必需配置：`hlr_service_center_gt_address` 参数是 强制性的。如果未配置，应用程序将无法启动。
- **OmniHSS** 后端：OmniHSS API 后端必须在配置的 `hlr_api_base_url` 处可访问
- **API** 请求超时：所有 OmniHSS API 请求都有一个 硬编码的 **5** 秒超时
- **MAP** 请求超时：所有 MAP 请求（SRI、UpdateLocation、SendAuthInfo 等）都有一个 硬编码的 **10** 秒超时
- **ISD** 超时：UpdateLocation 序列中的每个 InsertSubscriberData (ISD) 消息都有一个 硬编码的 **10** 秒超时
- 需要与 STP 建立 M3UA 连接以接收 MAP 操作
- 更改模式后，您必须重启应用程序以使更改生效
- **Web UI**：有关使用 Web 界面的信息，请参见 Web UI 指南
- **API** 访问：有关 REST API 文档和 Swagger UI 访问的信息，请参见 API 指南

用户数据库

OmniHSS 管理所有用户数据，包括身份、身份验证凭据、服务配置和位置信息。OmniSS7 通过 RESTful API 调用检索这些数据。

OmniHSS 用户模型

OmniHSS 存储全面的用户信息：

- 每个用户多个 **IMSI**：支持多 IMSI 配置（eSIM、漫游配置、网络切换）
- 身份验证凭据：Ki、OPc 和算法选择（Milenage 或 COMP128）
- 服务配置：用户类别、允许的服务、QoS 参数
- 位置跟踪：当前 VLR/MSC（电路会话）和 SGSN/GGSN（数据会话）独立跟踪
- **CAMEL** 订阅数据：服务密钥、触发点和 gsmSCF 地址
- 附加服务：呼叫转移、限制、等待、CLIP/CLIR 配置
- 管理状态：启用/禁用、服务限制、到期日期

身份验证向量

生成身份验证向量

OmniHSS 使用 **Milenage** 或 **COMP128** 算法生成身份验证向量，具体取决于每个用户配置的身份验证方法。当 OmniSS7 接收到 **sendAuthenticationInfo** MAP 请求时：

1. OmniSS7 从 MAP 请求中提取 IMSI
2. OmniSS7 调用 OmniHSS API 生成身份验证向量
3. OmniHSS 检索用户的 Ki 和 OPc 凭据
4. OmniHSS 生成请求数量的向量（RAND、XRES、CK、IK、AUTN）
5. OmniSS7 将向量编码为 MAP 格式并将其回给请求的 VLR/SGSN

OmniHSS API 集成

OmniSS7 通过 HTTPS REST API 与 OmniHSS 通信，以检索用户信息、更新位置信息和生成身份验证向量：

```
config :omniss7,
  hlr_api_base_url: "https://omnihss-server:8443"
```

当 OmniSS7 从 SS7 网络接收到 MAP 操作时，它查询 OmniHSS 以：

- 按 **IMSI** 或 **MSISDN** 检索用户数据
- 使用存储的 **Ki/OPc** 凭据生成身份验证向量
- 在用户执行 **UpdateLocation** 时更新电路会话位置
- 检查用户状态和服务权限

位置更新

更新位置处理

在接收到 **updateLocation** MAP 请求时，OmniSS7 与 OmniHSS 协调以在新的 VLR 注册用户：

1. 从 **UpdateLocation** 请求中提取位置信息（IMSI、新 VLR GT、新 MSC GT）
2. 查询 **OmniHSS** 以验证用户是否存在且已启用
3. 在 **OmniHSS** 中更新电路会话，以新的 VLR/MSC 位置
4. 发送 **InsertSubscriberData (ISD)** 消息以在新 VLR 中配置用户
5. 返回 **UpdateLocation** 响应 给 VLR（包括来自 `hlr_service_center_gt_address` 的 HLR GT）
6. 发送 **alertServiceCenter** 给配置的 SMSc GT（如果 `hlr_smsc_alert_gts` 被填充）

注意：`hlr_service_center_gt_address` 配置参数指定在 UpdateLocation 响应中□回的 HLR 的全球标题。这允许 VLR/MSC 识别并将消息路由回此 HLR。

警报服务中心集成

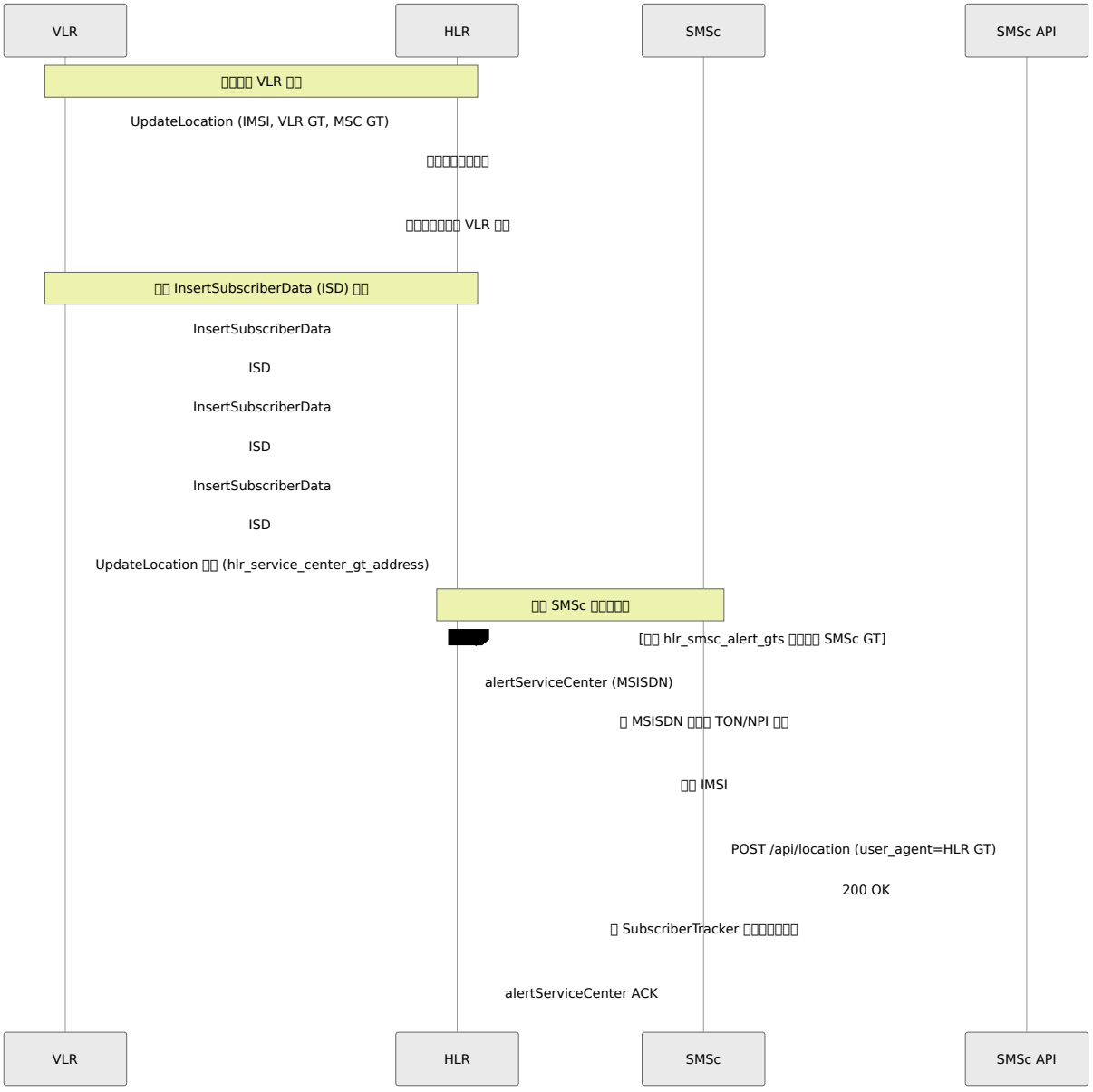
在成功的 UpdateLocation 之后，HLR 可以通过发送 **alertServiceCenter**（MAP 操作码 64）消息自动通知 SMSc 系统，用户现在可达。有关 SMSc 如何处理这些警报的信息，请参见 SMSc 指南中的警报服务中心处理。

配置

配置要通知的 SMSc 全球标题列表：

```
config :omniss7,
  # □ UpdateLocation □□□ alertServiceCenter □ SMSc GT □□
  hlr_smsc_alert_gts: [
    "15559876543",
    "15559876544"
  ],

  # SMSc □□□ alertServiceCenter □□□□□□□□□□□□□48 □□□
  hlr_alert_location_expiry_seconds: 172800
```

□□□



□□

□□□□□ UpdateLocation □□

1. HLR □ `hlr_smsc_alert_gts` □□□□ □□ SMSc GT □□ alertServiceCenter
2. □□□□□□□ MSISDN
3. HLR □□ `hlr_service_center_gt_address` □□□□□ GT
4. SCCP □□□□□ SSN=6□HLR□□□□ SSN=8□SMSc□

SMSc □□□□□□□

- □□ **MSISDN** □ **TON/NPI** □□□□□□“19123123213” → “123123213”□
- □□□□□□□□□□□□□□□□□□□ POST □ /api/location□
- □□□ **API** □□ `user_agent` □□□□□ **HLR GT**□□□□□□□ HLR □□□□□□
- □□ `hlr_alert_location_expiry_seconds` □□□□□□□□
- □ SMSc □□□□□□□□□□□□□□□□

□□

□□ Web UI □□ □□□□ □□□□□□ alertServiceCenter □□□□□□□

1. □□□“□□□□”□□□
2. □□“□□□□□□□□”□□
3. □□ MSISDN□SMSc GT □ HLR GT□□□□□□□□□□□□□
   - SMSc GT □□□□ `hlr_smsc_alert_gts` □□□□□□□
   - HLR GT □□□□ `hlr_service_center_gt_address`
4. □□“□□ alertServiceCenter”

□□□□□ SMSc □□□□□□□□□□□□□□□ UpdateLocation □□□□□□□□ `phx-blur` □□□□□□□□□□ □□□□

InsertSubscriberData (ISD) □□

□□□□ UpdateLocation □□□HLR □□ **InsertSubscriberData** (ISD) □□□□□□□□□□□□ VLR□ISD □□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□ □□□□□□□ ISD □□□

ISD □□

HLR 可以发送最多 3 条顺序 ISD 消息:

1. **ISD #1**（始终发送） - 基本用户数据:
    - IMSI
    - MSISDN
    - 用户类别
    - 用户状态（serviceGranted）
    - 承载服务列表
    - 电信服务列表
    - 网络访问模式
2. **ISD #2**（可选） - 附加服务（SS）数据:
    - 呼叫转移设置（无条件、忙、无回复、不可达）
    - 呼叫等待
    - 呼叫保持
    - 多方服务
    - 附加服务状态和功能
3. **ISD #3**（可选） - 呼叫限制数据:
    - 阻止所有外呼（BAOC）
    - 阻止外部国际呼叫（BOIC）
    - 访问限制数据

配置选项

```
# InsertSubscriberData 配置
# 网络访问模式: :packetAndCircuit、:packetOnly 或 :circuitOnly
isd_network_access_mode: :packetAndCircuit,

# 发送 ISD #2（附加服务数据）
isd_send_ss_data: true,

# 发送 ISD #3（呼叫限制数据）
isd_send_call_barring: true,
```

网络访问模式

`isd_network_access_mode` 参数控制用户被允许的网络访问类型:

值	描述	用例
`:packetAndCircuit`	同时支持分组交换（GPRS/LTE）和电路交换（语音）	默认 - 完整服务用户
`:packetOnly`	仅支持分组交换（数据/LTE）	数据卡、物联网设备
`:circuitOnly`	仅支持电路交换（语音/SMS）	旧设备、仅语音计划

控制 ISD 消息

您可以根据网络需求控制发送哪些 ISD 消息:

发送所有 **ISD**（默认 - 完整功能集）：

```
isd_send_ss_data: true,
isd_send_call_barring: true,
```

仅发送基本用户数据（最小配置）：

```
isd_send_ss_data: false,
isd_send_call_barring: false,
```

发送基本 **+** 附加服务（无呼叫限制）：

```
isd_send_ss_data: true,
isd_send_call_barring: false,
```

ISD 流程示例

当接收到 UpdateLocation 时:

```
VLR → HLR: UpdateLocation (□□)
HLR → VLR: InsertSubscriberData #1 (□□) - □□□□
VLR → HLR: ISD #1 ACK (□□)
HLR → VLR: InsertSubscriberData #2 (□□) - SS □□ [□□□□]
VLR → HLR: ISD #2 ACK (□□)
HLR → VLR: InsertSubscriberData #3 (□□) - □□□□ [□□□□]
VLR → HLR: ISD #3 ACK (□□)
HLR → VLR: UpdateLocation □□ (□□)
```

□□ `isd_send_ss_data` □ `isd_send_call_barring` □□□ `false`□□□□□□ ISD □□□□□□ □□ UpdateLocation □□□

□□□□

- □□□□□□□ `:packetAndCircuit` □□□□□ ISD□□□□□□□
- □□□/**M2M**□□□ `:packetOnly` □□□ SS □□/□□□□□□□□□□□□□□
- □□□□□□□□ VLR □□□□□□□□□□□ - □□□□□□□□□□ `isd_send_ss_data`
- □□□□□□□□□ ISD □□□□□□□□□□□□□□□□□

CAMEL □□

CAMEL □□□□ SendRoutingInfo

□□□□□ GMSC□□□ MSC□□ **SendRoutingInfo**□SRI□□□□□HLR □□□□ GMSC □□ CAMEL □□□□□□□□□□□□□□□□□

□□□□□□□□□□□□ CAMEL □□□□□□□□□

□□□ **CAMEL**□

CAMEL□□□□□□□□□□□□□□□□□□□□□□□□□□□□ GSM/UMTS □□□□□□□□□□□□□□□□□□□□

- □□□□
- □□□□□□□□□□□□□□
- □□□□□□□□

□□□□

```
# CAMEL □□□□□□ SendRoutingInfo □□□
# CAMEL □□□□□□□□□□
camel_service_key: 11_110,

# CAMEL □□□□□□
# □□□:termAttemptAuthorized□:tBusy□:tNoAnswer□:tAnswer
camel_trigger_detection_point: :termAttemptAuthorized,
```

□□□□

`camel_service_key` □□□ gsmSCF□□□□□□□□□□□□□□□□ CAMEL □□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□

□□□□	□□□□
`11_110`	□□□□□□□□□□□□□□□
`100`	□□□□□□□□
`200`	���□□□□□□□□□□□□
`300`	□□□□□□□□VPN□
□□□	□□□□□□□□

□□□□□

```
# 对于预付费终止呼叫控制
camel_service_key: 11_110,

# 对于 VPN 服务
camel_service_key: 300,
```

触发检测点

`camel_trigger_detection_point` 指定在呼叫设置期间何时触发 CAMEL 服务:

检测点	描述	何时触发
`:termAttemptAuthorized`	呼叫尝试已授权（默认）	在呼叫路由到用户之前
`:tBusy`	终止忙碌	当用户忙时
`:tNoAnswer`	终止未应答	当用户未应答时
`:tAnswer`	终止应答	当用户接听电话时

配置示例:

标准预付费控制（在路由之前触发）：

```
camel_trigger_detection_point: :termAttemptAuthorized,
```

自定义忙碌处理（在忙碌时触发）：

```
camel_trigger_detection_point: :tBusy,
```

基于应答计费（在应答时触发）：

```
camel_trigger_detection_point: :tAnswer,
```

带 CAMEL 的 SRI 响应

配置后，SendRoutingInfo 响应将包括 CAMEL 订阅信息：

```
GMSC → HLR: SendRoutingInfo (开始)
HLR → GMSC: SRI 响应 (结束) 包含:
  - IMSI
  - VLR 编号
  - 用户状态
  - CAMEL 路由信息:
    * 服务密钥: 11_110
    * gsmSCF 地址: <配置的地址>
    * 触发检测点: termAttemptAuthorized
    * 默认呼叫处理: continueCall

GMSC 在触发点联系 gsmSCF 执行 CAMEL 服务
```

最佳实践

- 生产网络：使用与您的 gsmSCF 提供商达成一致的标准化服务密钥
- 测试：使用 `:termAttemptAuthorized` 进行最全面的测试
- 预付费服务：服务密钥 `11_110` 是预付费终止呼叫的常见行业标准
- 回退处理：`defaultCallHandling: :continueCall` 确保在 gsmSCF 无法访问时呼叫继续进行

漫游用户处理

家庭 **VLR** 与漫游 **VLR** 检测

当 HLR 接收到 **SendRoutingInfo**（SRI）请求时，它需要确定用户是在“家庭” VLR（在您的网络内）还是在漫游 VLR（访问其他网络）。根据此确定，行为有所不同：

有关配置参数参考，请参见 配置参考中的家庭 VLR 前缀。

- 家庭 **VLR**：□回带有 CAMEL 路由信息的标准 SRI 响应
- 漫游 **VLR**：发送提供漫游号码（PRN）请求以获取 MSRN，然后在 SRI 响应中□回

□□

```
# □□ VLR □□
# □□□“□□”□□□ VLR □□□□□□□
# □□□□□□ VLR □□□□□□□
```

```
GMSC → HLR: SendRoutingInfo (MSISDN: "1234567890")
HLR 查询后端 API 获取用户数据
HLR 检查 VLR 地址: "49170123456"
HLR 确定: VLR 不以 "555123" 开头 → 漫游
HLR → MSC: ProvideRoamingNumber (PRN):
  - MSISDN: "1234567890"
  - IMSI: "999999876543210"
  - MSC 编号: "49170123456"
  - GMSC 地址: "5551234501"
MSC → HLR: PRN 响应带有 MSRN: "49170999888777"
HLR → GMSC: SRI 响应带有路由信息:
  - IMSI
  - VLR 编号: "49170123456"
  - 漫游号码 (MSRN): "49170999888777"
```

响应结构差异

家庭用户 **SRI** 响应

```
%{
  imsi: "999999876543210",
  extendedRoutingInfo: {
    :camelRoutingInfo, %{
      gmscCamelSubscriptionInfo: %{
        "t-CSI": %{
          serviceKey: 11_110,
          "gsmSCF-Address": "5551234501",
          defaultCallHandling: :continueCall,
          "t-BcsmTriggerDetectionPoint": :termAttemptAuthorized
        }
      }
    }
  },
  subscriberInfo: %{
    locationInformation: %{"vlr-number": "5551234567"},
    subscriberState: {:notProvidedFromVLR, :NULL}
  }
}
```

漫游用户 **SRI** 响应

```
%{
  imsi: "999999876543210",
  extendedRoutingInfo: {
    :routingInfo, %{
      roamingNumber: "49170999888777"  # 从 PRN 获取的 MSRN
    }
  },
  subscriberInfo: %{
    locationInformation: %{"vlr-number": "49170123456"},
    subscriberState: {:notProvidedFromVLR, :NULL}
  }
}
```

提供漫游号码（**PRN**）操作

PRN 请求结构

发送到 MSC/VLR 的 PRN 请求包含：

字段	来源	描述
MSISDN	SRI 请求	用户的电话号码
IMSI	HLR API	用户的 IMSI
MSC 编号	HLR API	为漫游用户提供服务的 MSC（`serving_msc`）
GMSC 地址	SRI 请求	发起原始 SRI 请求的 GMSC
呼叫参考编号	静态	呼叫参考标识符
支持的 **CAMEL** 阶段	静态	GMSC 支持的 CAMEL 阶段

PRN 响应处理

HLR 期望 PRN 响应包含：

- **MSRN**（移动台漫游号码）：由访问网络分配的临时号码，用于路由呼叫

错误处理：

- 如果 PRN 超时 → 在 SRI 响应中回错误 27（缺少用户）
- 如果 PRN 失败 → 在 SRI 响应中回错误 27（缺少用户）
- 如果无法提取 MSRN → 在 SRI 响应中回错误 27（缺少用户）

配置示例

单一家庭网络运营商

```
# 所有 VLR 地址以 "555123" 开头被视为家庭
home_vlr_prefixes: ["555123"],
```

- VLR `5551234567` → 家庭（CAMEL 响应）
- VLR `5551235001` → 家庭（CAMEL 响应）
- VLR `49170123456` → 漫游（PRN + MSRN 响应）

多区域运营商

```
# 跨不同区域的多个家庭网络
home_vlr_prefixes: ["555123", "555124", "555125"],
```

- VLR `5551234567` → 家庭（区域 1）
- VLR `5552341234` → 家庭（区域 2）
- VLR `5553411111` → 家庭（区域 3）
- VLR `44201234567` → 漫游（国际）

测试配置

为了测试 PRN 功能，设置一个空列表以将所有 VLR 视为漫游：

```
# 所有 VLR 被视为漫游（用于测试 PRN 流程）
home_vlr_prefixes: [],
```

最佳实践

- 前缀选择：使用唯一的最短前缀来识别您网络的 VLR（例如，国家代码 + 网络代码）
- 多个前缀：包括您网络中的所有 VLR 前缀，包括不同区域和子公司
- 漫游协议：确保漫游合作网络正确支持 PRN
- 测试：在生产部署之前彻底测试家庭和漫游场景
- 监控：监控 PRN 超时率，以识别与漫游合作伙伴的连接问题

故障排除

症状：所有用户都被视为漫游

- 原因：未配置 `home_vlr_prefixes` 或前缀与 VLR 地址不匹配
- 解决方案：检查数据库中的 VLR 地址并相应更新前缀

症状：PRN 请求超时

- 原因：与漫游合作伙伴 MSC/VLR 的网络连接问题
- 解决方案：验证与远程 MSC 地址的 M3UA/SCCP 路由

症状：SRI 响应中的 MSRN 无效

- 原因：漫游合作伙伴的 PRN 响应格式与预期结构不匹配
- 解决方案：检查 PRN 响应日志，并根据需要调整 `extract_msrn_from_prn/1`

HLR 操作

支持的 MAP 操作

- `updateLocation`（操作码 2）- 注册 VLR 位置
- `sendAuthenticationInfo`（操作码 56）- 生成身份验证向量
- `sendRoutingInfo`（操作码 22）- 为支持 CAMEL 的呼叫提供 MSRN
- `sendRoutingInfoForSM`（操作码 45）- 为 SMS 提供 MSC GT
- `cancelLocation`（操作码 3）- 从旧 VLR 注销

- `insertSubscriberData`（操作码 7）- 推送用户配置

响应字段映射

本节详细说明 HLR 响应中每个字段的来源。

SendRoutingInfo (SRI) 响应

目的：提供用户的呼叫路由信息。

HLR 根据用户是在家庭 VLR 还是漫游提供两种不同的响应类型：

家庭用户响应（CAMEL 路由）

用于用户的 VLR 地址以配置的 `home_vlr_prefixes` 值开头时。

响应结构：

字段	来源	描述
IMSI	OmniHSS API	来自 OmniHSS 数据库的用户 IMSI
VLR 编号	OmniHSS API	当前为用户提供服务的 VLR（`circuit_session.assigned_vlr`）
用户状态	静态	始终为 `notProvidedFromVLR`
extendedRoutingInfo	-	类型：`camelRoutingInfo`
gsmSCF 地址	OmniHSS API	为用户提供服务的 MSC（`circuit_session.assigned_msc`）
服务密钥	runtime.exs	CAMEL 服务标识符（`camel_service_key`）
触发检测点	runtime.exs	何时触发 CAMEL（`camel_trigger_detection_point`）
CAMEL 能力处理	静态	CAMEL 阶段支持级别
默认呼叫处理	静态	如果 gsmSCF 无法访问的回退

漫游用户响应（**MSRN** 路由）

用于用户的 VLR 地址不匹配任何配置的 `home_vlr_prefixes` 值时。

响应结构：

字段	来源	描述	
IMSI	OmniHSS API	来自 OmniHSS 数据库的用户 IMSI	"
VLR 编号	OmniHSS API	当前为用户提供服务的 VLR（`circuit_session.assigned_vlr`）	"
用户状态	静态	始终为 `notProvidedFromVLR`	:
extendedRoutingInfo	-	类型: `routingInfo`	-
漫游号码（**MSRN**）	PRN 响应	从提供漫游号码请求中获得的 MSRN	"

路由决策逻辑:

```
1. OmniSS7 接收到 SendRoutingInfo 请求
2. OmniSS7 查询 OmniHSS API 获取用户数据
3. OmniSS7 检查 VLR 地址与 home_vlr_prefixes:

   如果 VLR 以家庭前缀开头:
     → 返回 CAMEL 路由信息（家庭用户流）

   如果 VLR 不匹配何家庭前缀:
     → 向 MSC 发送提供漫游号码（PRN）
     → 从 PRN 响应中提取 MSRN
     → 返回带有 MSRN 的路由信息（漫游用户流）
```

数据流:

- OmniSS7 查询 OmniHSS 获取用户信息
- OmniHSS 回 IMSI、当前 VLR/MSC 位置和用户状态
- OmniSS7 使用这些数据构建 MAP 响应

配置要求:

```
# 在 runtime.exs 中
home_vlr_prefixes: ["555123"],  # 家庭 VLR 前缀列表
```

错误响应：

- 如果 `serving_vlr` 和 `serving_msc` 为 `null`：回错误 27（缺少用户）
- 如果未找到用户：回错误 1（未知用户）
- 如果 PRN 请求超时（漫游情况）：回错误 27（缺少用户）
- 如果 PRN 响应无效（漫游情况）：回错误 27（缺少用户）

UpdateLocation 响应与 InsertSubscriberData

目的：在新 VLR 中注册用户并配置用户数据。

UpdateLocation 结束响应

字段	来源	描述	示例
HLR 编号	runtime.exs	此 HLR 的全球标题（`hlr_service_center_gt_address`）	`"5551234568"`
TCAP 消息类型	静态	在所有 ISD 之后的最终响应	`END`

InsertSubscriberData #1（基本用户数据）

字段	来源	描述	示例
IMSI	请求	来自 UpdateLocation 请求	`"999999876543210`
MSISDN	OmniHSS API	来自 OmniHSS 的用户电话号码	`"555123456"`
类别	静态	用户类别	`"\n"` (0x0A)
用户状态	静态	服务状态	`:serviceGranted`
承载服务列表	静态	支持的承载服务	`[<<31>>]`
电信服务列表	静态	支持的电信服务	`[<<17>>, "!", "\""]`
网络访问模式	runtime.exs	分组/电路访问（`isd_network_access_mode`）	`:packetAndCircuit`

InsertSubscriberData #2（附加服务） - 可选

字段	来源	描述	由谁控制
配置的 **SS**	静态	附加服务数据	`isd_send_ss_data: true`
呼叫转移	静态	转移配置（无条件、忙、无回复、不可达）	配置启用
呼叫等待	静态	呼叫等待服务状态	配置启用
多方服务	静态	会议呼叫支持	配置启用

InsertSubscriberData #3（呼叫限制） - 可选

字段	来源	描述	由谁控制
呼叫限制信息	静态	呼叫限制配置	`isd_send_call_barring: true`
BAOC	静态	阻止所有外呼（SS 代码 146）	配置启用
BOIC	静态	阻止外部国际呼叫（SS 代码 147）	配置启用
访问限制数据	静态	网络访问限制	配置启用

ISD 序列控制：

- ISD #1：始终发送 - 包含基本用户数据
- ISD #2：仅在 `isd_send_ss_data: true` 时发送
- ISD #3：仅在 `isd_send_call_barring: true` 时发送

SendRoutingInfoForSM (SRI-for-SM) 响应

目的：提供 SMS 交付的 MSC/SMSC 路由信息。当 SMSc 需要将 SMS 发送到用户时，它向 HLR 发送 SRI-for-SM 请求以确定消息的路由位置。

响应结构：

字段	来源	描述	如何生成	
IMSI	计算	从 MSISDN 派生的合成 IMSI	`PLMN_PREFIX + zero_padded_MSISDN`	`"00100`
网络节点编号	runtime.exs	用于 SMS 路由的 SMSC GT 地址	`smsc_service_center_gt_address`	`"55512`

配置参数（来自 `runtime.exs`）：

```
# 服务中心 GT 地址（在 SRI-for-SM 响应中返回）
# 这告诉请求的 SMSc 应该将 MT-ForwardSM 消息发送到哪里
smsc_service_center_gt_address: "5551234567",   # 必需

# MSISDN ↔ IMSI 映射配置
# PLMN 前缀：MCC（001 = 测试网络）+ MNC（01 = 测试运营商）
hlr_imsi_plmn_prefix: "001001",                          # 唯一需要的配置参数！
```

MSISDN ↔ IMSI 映射

配置参数：

这些参数控制 OmniSS7 如何为 SRI-for-SM 响应生成合成 IMSI：

- **`hlr_imsi_plmn_prefix`**：用于构建合成 IMSI 的 MCC+MNC 前缀（例如，“50557”表示 MCC=505，MNC=57）
- **`hlr_msisdn_country_code`**：在进行反向 IMSI→MSISDN 映射时要添加的国家代码（例如，“61”表示澳大利亚，“1”表示美国/加拿大）
- **`hlr_msisdn_nsn_offset`**：MSISDN 中国家用户号码（NSN）开始的字符位置（通常为 0，如果 MSISDN 不包括国家代码，或者为国家代码的长度）
- **`hlr_msisdn_nsn_length`**：要从 MSISDN 中提取的 NSN 的位数

有关更多配置详细信息，请参见 配置参考中的 MSISDN ↔ IMSI 映射。

为什么需要 **MSISDN** 到 **IMSI** 的映射？

SendRoutingInfoForSM（SRI-for-SM）协议要求 HLR 在响应中□回 **IMSI**（国际移动用户身份）。然而，请求的 SMSc 只知道用户的 **MSISDN**（电话号码）。

在传统网络中：

- SMSc 发送带有目标 MSISDN 的 SRI-for-SM（例如，“5551234567”）
- HLR 必须在其数据库中查找用户以找到他们的 IMSI
- HLR 在 SRI-for-SM 响应中□回 IMSI
- SMSc 然后在将 MT-ForwardSM ���送到 MSC/VLR 时使用此 IMSI

OmniSS7 的方法 **-** 合成 **IMSI**：

OmniSS7 □□□□□□□□□□□□□□□ MSISDN □ IMSI □□□□□□□□□□□□□□□□□□□□□□□□□□□□ IMSI□□□□□□□□□□□□□□□□

1. □□□□□□□ HLR □□□□□□□□□□ IMSI □□□□□□□□ SS7 □□□□□□ SRI-for-SM □□□□□□
2. □□□□□□ SRI-for-SM □□□□□□□□□□ HLR □□□□□□ IMSI □□ - IMSI □□□ MSISDN □□□□□□

□□□□□□

MSISDN □□□□□□ IMSI □□□□□□□MCC+MNC □□□□□□□□

```
IMSI = PLMN_PREFIX + zero_padded_MSISDN
```

□□□

- **PLMN_PREFIX**□MCC + MNC□□□□“001001”□□□□□□□□
- **MSISDN**□□□□□□□□□□□□□
- □□□□□□□□□□□□ IMSI □□□ 15 □

□□□□□□

```
# □□
plmn_prefix = "001001"  # MCC 001 + MNC 01

# □□□□□□ SRI-for-SM □□□ MSISDN□TBCD □□□
msisdn = "555123456"    # 9 □□□

# □□ 1□□□□□□□□□□□□□□
subscriber_digits = 15 - String.length("001001")  # = 9 □□□

# □□ 2□□□□□□ MSISDN □□□□□□□□
padded_msisdn = String.pad_leading("555123456", 9, "0")  # =
"555123456"□□□□□□□

# □□ 3□□□ PLMN □□ + □□□ MSISDN
imsi = "001001" <> "555123456"  # = "001001555123456"□□□ 15 □□
```

□□□□□□

输入 **MSISDN**	**PLMN** 前缀	可用的用户数字	填充的 **MSISDN**	最终 **IMSI**	备注
`"555123456"`	`"001001"` (6)	9	`"555123456"`	`"001001555123456"`	完全适合，无需填充
`"99"`	`"001001"` (6)	9	`"000000099"`	`"001001000000099"`	左侧填充零
`"999999999"`	`"001001"` (6)	9	`"999999999"`	`"001001999999999"`	完全适合
`"91123456789"`	`"001001"` (6)	9	`"555123456"`	`"001001555123456"`	太长，保留右侧 9 位

边缘情况处理：

- 短 **MSISDN**：左侧填充零（例如，“99” → “000000099”）
- 长 **MSISDN**：保留右侧数字，截断左侧数字（例如，“91123456789” → “555123456”）
- **IMSI** 长度：始终恰好为 15 位

反向映射（**IMSI → MSISDN**）：

SMSc 可以反向此映射将 IMSI 转换回 MSISDN：

```
# □□□□□□ SRI-for-SM □□□ IMSI
imsi = "001001555123456"

# □□ 1□□□ PLMN □□
plmn_prefix = "001001"
subscriber_portion = String.slice(imsi, 6, 9)  # = "555123456"

# □□ 2□□□□□□□□□□□□□□□ MSISDN
msisdn = String.replace_leading(subscriber_portion, "0", "")  # =
"555123456"
```

□□□□□□□□

□□ IMSI	PLMN □□	□□□□	□□□□□	□□ MSISD
`"001001555123456"`	`"001001"`	`"555123456"`	`"555123456"`	`"5551234`
`"001001000000099"`	`"001001"`	`"000000099"`	`"99"`	`"99"`
`"001001999999999"`	`"001001"`	`"999999999"`	`"999999999"`	`"9999999`

□□□□□□□□

- □ □□□□□□□□ MSISDN □□□□□□□□ IMSI
- □ □□□□□□□□ IMSI □□□ MSISDN
- □ □□□□□□□□□ `hlr_imsi_plmn_prefix`
- □ □□□□□□□□ IMSI □□□□
- □ □□□□□□□□□ HLR □□□
- □ □□ **15** □□IMSI □□□□□ 15 □

MSISDN □□□□□

□ HLR □□□ SRI-for-SM □□□□MSISDN □□□ TBCD □□□

1. **TBCD** □□□□□□□□ TBCD □□□□□□□□□□□□ TON/NPI □□□□“91”□
2. □□□□□□□□□□□□□□□□□□□□□□□□
3. □□□□□□□□□□□□□□□□□□□□

4. 编码：连接 PLMN 前缀 + 标准化[illegible] MSISDN

安全考虑：

在 SRI-for-SM 响应中[illegible]回的合成 IMSI 仅用于路由目的。它们不是存储在 HLR 用户数据库中的真实 IMSI。这提供了额外的隐私保护层，因为真实用户 IMSI 仅在绝对必要时暴露（例如，在 UpdateLocation 或 SendAuthenticationInfo 操作中需要真实的身份验证向量）。

响应流：

```
1. SMSc → HLR: SRI-for-SM 请求
   - MSISDN（TBCD）："91123456789"（包括 TON/NPI）

2. HLR 处理：
   - TBCD 解码："91123456789"
   - 提取数字："91123456789"（11 位数字）
   - 填充到 9 位数字："555123456"（保留右侧 9 位）
   - 添加 PLMN："001001" + "555123456" = "001001555123456"
   - 从配置中获取 SMSC GT："5551234567"

3. HLR → SMSc: SRI-for-SM 响应
   - IMSI："001001555123456"（合成，始终 15 位）
   - 网络节点编号："5551234567"（发送 MT-ForwardSM 的位置）

4. SMSc 将 MT-ForwardSM 发送到 "5551234567"，IMSIN 为 "001001555123456"
```

配置：

以下参数在 `runtime.exs` 中使用：

```
# PLMN 前缀：MCC（001 = 测试网络）+ MNC（01 = 测试运营商）
hlr_imsi_plmn_prefix: "001001",

# NSN 提取（如果 MSISDN 包括国家代码）
hlr_msisdn_country_code: "1",       # 用于反向映射（IMSI→MSISDN）
hlr_msisdn_nsn_offset: 1,           # 跳过 1 位国家代码
hlr_msisdn_nsn_length: 10           # 提取 10 位 NSN
```

NSN 提取配置：

如果您的 MSISDN 包括国家代码（例如，“68988000088”而不是仅仅“88000088”），则必须配置 NSN 提取：

- `hlr_msisdn_nsn_offset`：NSN 开始的位置（通常为国家代码的长度）
- `hlr_msisdn_nsn_length`：要从 MSISDN 中提取的 NSN 位数

示例：

示例	国家代码	**MSISDN** 示例	**nsn_offset**	**nsn_length**	提取的 **NSN**
1 位 CC	`"9"`	`"95551234567"`	`1`	`10`	`"5551234567"`
2 位 CC	`"99"`	`"99412345678"`	`2`	`9`	`"412345678"`
3 位 CC	`"999"`	`"99988000088"`	`3`	`8`	`"88000088"`

工作原理：

1. **MSISDN → IMSI**：从 MSISDN 中提取 NSN，填充零，连接 PLMN 前缀

```
MSISDN: "99988000088"
NSN: String.slice("99988000088", 3, 8) = "88000088"
填充的 NSN: "088000088"（9 位数字）
IMSI: "547050" + "088000088" = "547050088000088"
```

2. **IMSI → MSISDN**：剥离 PLMN 前缀，去除前导零，添加国家代码

```
IMSI: "547050088000088"
用户部分: "088000088"
去除零: "88000088"
MSISDN: "+999" + "88000088" = "+99988000088"
```

API 要求：无 - SRI-for-SM 使用计算值和配置。无需后端 API 调用。

字段源摘要

源类型	描述	示例
OmniHSS API	来自 OmniHSS 用户数据库的动态数据	IMSI、MSISDN、来自 circuit_session 的服务 VLR/MSC
runtime.exs	OmniSS7 配置参数	`smsc_service_center_gt_address`、`camel_service_key`、`isd_network_access_mode`
静态	响应生成器中的硬编码值	用户状态、承载服务、SS 代码
请求	从传入 MAP 请求中提取的字段	来自 UpdateLocation 的 IMSI、来自 SRI 的 MSISDN
计算	使用逻辑生成的派生值	SRI-for-SM 中的合成 IMSI（hlr_imsi_prefix + NSN）

配置依赖关系

在 **runtime.exs** 中必需：

- `hlr_service_center_gt_address` - 在 UpdateLocation 响应中使用
- `smsc_service_center_gt_address` - 在 SRI-for-SM 响应中使用（MT-ForwardSM 应路由到的位置）

在 **runtime.exs** 中可选（带默认值）：

- `camel_service_key` - 默认：`11_110`
- `camel_trigger_detection_point` - 默认：`:termAttemptAuthorized`

- `isd_network_access_mode` - 默认: `:packetAndCircuit`
- `isd_send_ss_data` - 默认: `true`
- `isd_send_call_barring` - 默认: `true`
- `hlr_imsi_plmn_prefix` - 默认: `"001001"`（用于 MSISDN↔IMSI 映射的 PLMN 前缀）

来自 **OmniHSS** 的必需:

OmniHSS 必须提供 REST API 端点以:

- 按 IMSI 和 MSISDN 查询用户
- 更新电路会话位置（VLR/MSC 分配）
- 生成身份验证向量
- 查询用户状态和服务配置

相关文档

OmniSS7 文档:

- ← 回主文档
- 常见功能指南
- MAP 客户端指南
- 技术参考
- 配置参考

OmniHSS 文档：有关用户管理、配置、身份验证配置和管理操作的信息，请参阅 **OmniHSS** 产品文档。OmniHSS 包含所有用户数据库逻辑、身份验证算法、服务配置规则和多 IMSI 管理能力。

OmniSS7 由 Omnitouch 网络服务提供

MAP客户端配置□南

← □回主文档

本指南提供了使用OmniSS7作为**MAP**客户端向网络元素发送MAP协议请求的详细配置。

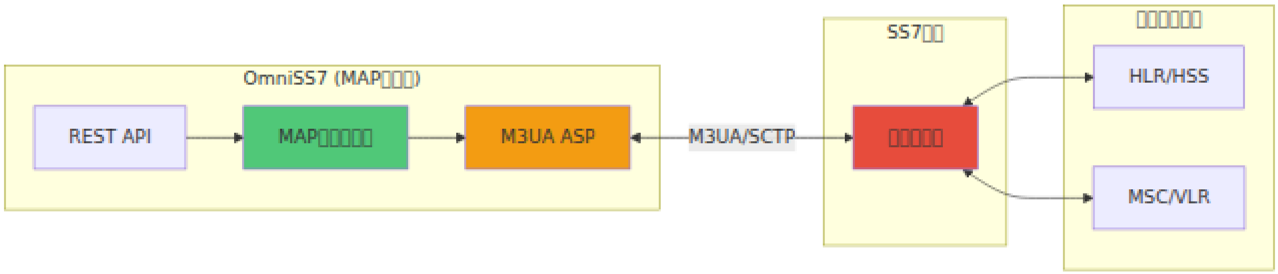
目录

什么是**MAP**客户端模式?

MAP客户端模式允许OmniSS7作为**应用服务器进程（ASP）连接到**M3UA**对等体（**STP**或**SGP**），并发送/接收MAP（移动应用部分）**消息，用于以下服务：

- **HLR**查询：SRI（发送路由信息）、SRI-for-SM、认证信息
- 位置更新：更新位置、取消位置
- 用户管理：提供漫游号码（PRN）、插入用户数据

网络架构



启用**MAP**客户端模式

编辑`config/runtime.exs`并配置MAP客户端设置。有关完整的配置参考，请参见配置参考中的M3UA连接参数。

基本配置

```
config :omniss7,
  # 启用MAP客户端模式
  map_client_enabled: true,

  # MAP客户端的M3UA连接（作为ASP连接到远程STP/SGP）
  map_client_m3ua: %{
    mode: "ASP",                      # M3UA模式："ASP"（客户端）或"SGP"（服务器）
    callback: {MapClient, :handle_payload, []},  # 处理传入消息的回调
    process_name: :map_client_asp,    # 注册的进程名称
    local_ip: {10, 0, 0, 100},       # 本地IP地址
    local_port: 2905,                 # 本地SCTP端口
    remote_ip: {10, 0, 0, 1},        # 远程STP/SGP IP
    remote_port: 2905,                # 远程STP/SGP端口
    routing_context: 1                # M3UA路由上下文
  }
```

生产配置示例

```
config :omniss7,
  # 为生产启用MAP客户端
  map_client_enabled: true,

  # 生产M3UA连接
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :map_client_asp,
    local_ip: {10, 0, 0, 100},
    local_port: 2905,
    remote_ip: {10, 0, 0, 1},        # 生产STP IP
    remote_port: 2905,
    routing_context: 1
  }

config :control_panel,
  web: %{
    listen_ip: "0.0.0.0",
    port: 443,
    hostname: "ss7-gateway.example.com",
    enable_tls: true,
    tls_cert: "/etc/ssl/certs/gateway.crt",
    tls_key: "/etc/ssl/private/gateway.key"
  }
```

可用的MAP操作

1. 为SM发送路由信息（SRI-for-SM）

查询HLR以确定SMS投递的服务MSC。有关HLR如何处理SRI-for-SM请求的详细信息，请参见HLR指南中的SRI-for-SM。

API□□□ `POST /api/sri-for-sm`

□□□

```json
{
  "msisdn": "447712345678",
  "serviceCenter": "447999123456"
}
```

□□□

```json
{
  "result": {
    "imsi": "234509876543210",
    "locationInfoWithLMSI": {
      "networkNode-Number": "447999555111"
    }
  }
}
```

cURL□□□

```
curl -X POST http://localhost/api/sri-for-sm \
  -H "Content-Type: application/json" \
  -d '{
    "msisdn": "447712345678",
    "serviceCenter": "447999123456"
  }'
```

---

2. □□□□□□□SRI□

□□HLR□□□□□□□□□□□□

API□□□ `POST /api/sri`

□□□

```
{
  "msisdn": "447712345678",
  "gmsc": "447999123456"
}
```

响应:

```
{
  "result": {
    "imsi": "234509876543210",
    "extendedRoutingInfo": {
      "routingInfo": {
        "roamingNumber": "447999555222"
      }
    }
  }
}
```

---

3. 提供漫游号码（PRN）

请求服务MSC的临时漫游号码（MSRN）。

API端点: `POST /api/prn`

请求:

```
{
  "msisdn": "447712345678",
  "gmsc": "447999123456",
  "msc_number": "447999555111",
  "imsi": "234509876543210"
}
```

---

4. 发送认证信息

请求HLR的认证向量以进行用户认证。

API端点: `POST /api/send-auth-info`

请求:

```
{
  "imsi": "234509876543210",
  "vectors": 5
}
```

响应:

```
{
  "result": {
    "authenticationSetList": [
      {
        "rand": "0123456789ABCDEF0123456789ABCDEF",
        "xres": "ABCDEF0123456789",
        "ck": "0123456789ABCDEF0123456789ABCDEF",
        "ik": "FEDCBA9876543210FEDCBA9876543210",
        "autn": "0123456789ABCDEF0123456789ABCDEF"
      }
    ]
  }
}
```

---

5. 更新位置

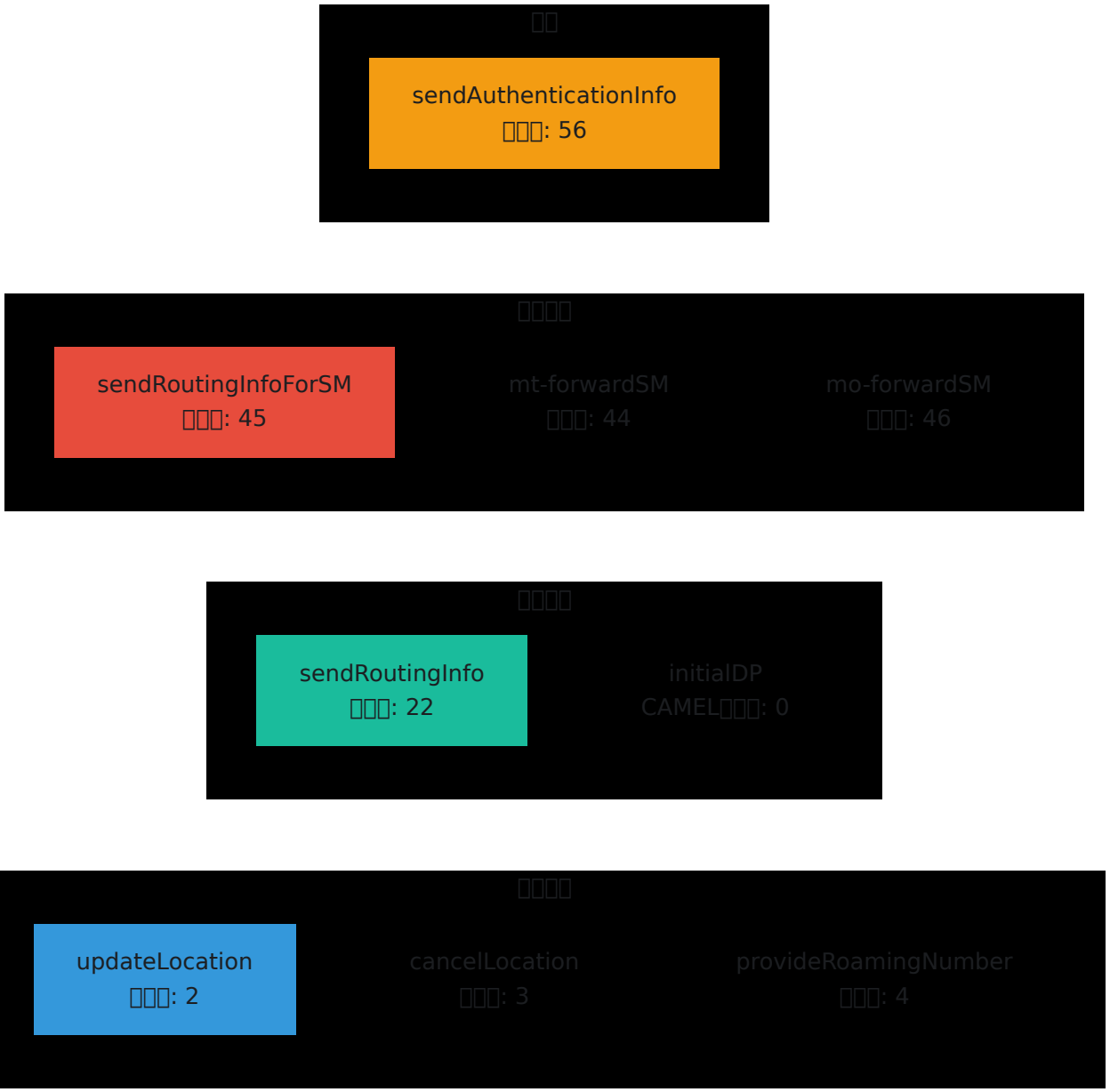
向HLR注册用户的当前位置。有关更新位置处理和插入用户数据序列的详细信息，请参见HLR指南中的位置更新。

API端点: `POST /api/updateLocation`

请求:

```
{
  "imsi": "234509876543210",
  "vlr": "447999555111"
}
```

MAP□□□□



通过API发送请求

使用Swagger UI

Swagger UI提供了一个交互式界面，用于发送SS7请求。

访问**Swagger UI**:

1. 导航到`http://your-server/swagger`
2. 浏览可用的API端点
3. 点击任何端点以展开其详细信息

发送请求:

1. 点击您想使用的端点（例如，`/api/sri-for-sm`）
2. 点击“试用”按钮
3. 在请求体中填写所需参数
4. 点击“执行”
5. 查看下面的响应

API响应代码

- **200** - 成功，结果回在响应体中
- **400** - 错误请求，参数无效
- **504** - 网关超时，SS7网络在10秒内没有响应

MAP客户端标

可用标

请求标:

- `map_requests_total` - 发送的MAP请求总数
  - 标签: `operation` (值: `sri`, `sri_for_sm`, `prn`, `authentication_info`等)

- `map_request_errors_total` - MAP请求错误的总数
  - 标签: `operation`
- `map_request_duration_milliseconds` - MAP请求持续时间的直方图
  - 标签: `operation`
- `map_pending_requests` - 当前待处理的MAP请求数量（仪表）

示例**Prometheus**查询

```
# 最近一小时内的总SRI-for-SM请求
increase(map_requests_total{operation="sri_for_sm"}[1h])

# SRI请求的平均响应时间
rate(map_request_duration_milliseconds_sum{operation="sri"}[5m]) /
rate(map_request_duration_milliseconds_count{operation="sri"}[5m])

# 所有MAP操作的错误率
sum(rate(map_request_errors_total[5m])) by (operation)

# 当前待处理请求
map_pending_requests
```

故障排除**MAP**客户端

问题：请求超时

症状:

- API回504网关超时
- HLR/MSC没有响应

检查:

1. 验证M3UA连接是否处于活动状态:

```
# 在IEx控制台中
:sys.get_state(:map_client_asp)
```

2. 检查与STP的网络连接
3. 验证路由上下文和SCCP寻址
4. 检查日志中的SCCP错误

问题：SCCP错误

症状：

- API返回SCCP错误响应
- 日志显示“SCCP unitdata service”消息

常见**SCCP**错误代码：

- 无翻译：在STP路由表中找不到全局标题
- 子系统故障：目标子系统（HLR SSN 6）不可用
- 网络故障：网络拥塞或故障

解决方案：

- 联系STP管理员以验证路由配置
- 验证目标全局标题是否可达
- 检查目标子系统是否正常运行

相关文档

- ← 返回主文档
- 常见功能指南 - Web UI、API、监控
- STP指南 - 路由配置
- 短信中心指南 - 短信投递

- 技术参考 - 协议规范

OmniSS7由Omnitouch网络服务提供

短信中心 (SMSc) 配置指南

← 返回主文档

本指南提供了使用 OmniSS7 作为 短信中心 (SMSc) 前端与 OmniMessage 作为后端消息存储和传递平台的详细配置。

OmniMessage 集成

OmniSS7 SMSc 模式作为 **SS7** 信令前端，与 **OmniMessage** 进行接口，这是一个运营商级的 SMS 平台。该架构分离了关注点：

- **OmniSS7 (SMSc 前端)**：处理所有 SS7/MAP 协议信令、SCCP 路由和网络通信
- **OmniMessage (SMS 后端)**：管理消息存储、排队、重试逻辑、交付跟踪和路由决策

为什么选择 OmniMessage？

OmniMessage 提供运营商级的 SMS 消息传递能力，具有以下功能：

- 消息队列管理：持久存储，具有可配置的重试逻辑和优先级排队
- 交付跟踪：实时交付状态、交付报告 (DLR) 和失败原因跟踪
- 多 **SMSc** 支持：多个前端实例可以连接到单个 OmniMessage 后端以实现负载均衡和冗余
- 路由智能：基于目的地、发送者、消息内容和时间的高级路由规则
- 速率限制：每条路由的 TPS（每秒事务）控制，以防止网络拥塞
- **API** 优先设计：RESTful HTTP API 用于与计费系统、客户门户和第三方应用集成
- 分析与报告：消息量统计、交付成功率和性能指标

所有消息数据、交付状态和路由配置都存储和管理在 OmniMessage 中。OmniSS7 通过 HTTPS API 调用查询 OmniMessage，以检索待处理消息、更新交付状态并注册为活动前端。

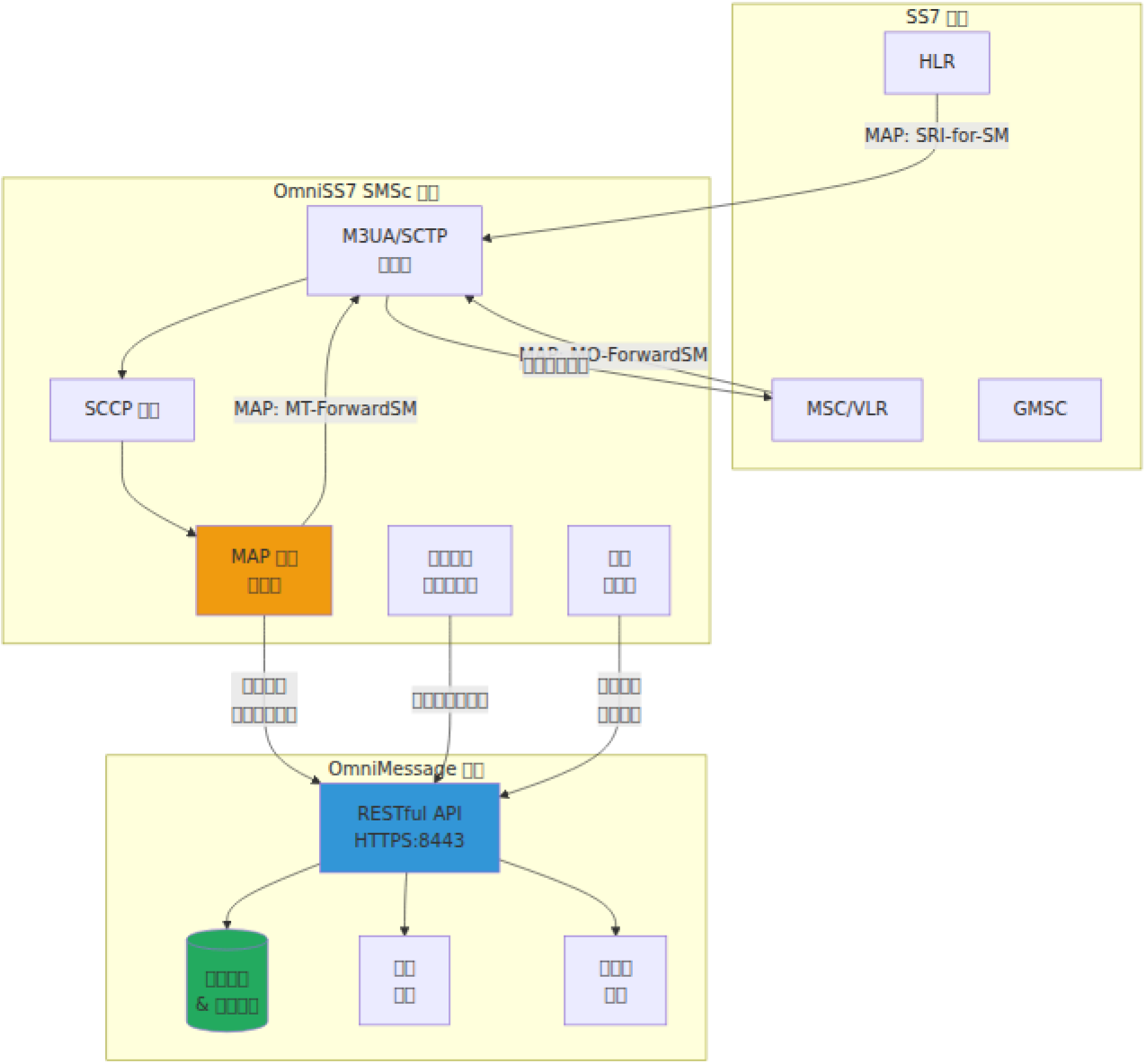
重要：OmniSS7 SMSc 模式仅为 信令前端。所有消息路由逻辑、队列管理、重试算法、交付跟踪和业务规则均由 OmniMessage 处理。本指南涵盖 OmniSS7 中的 SS7/MAP 协议配置。有关消息路由、队列配置、交付报告、速率限制和分析的信息，请 参考 **OmniMessage** 文档。

□□

□□□□□□□□□□

□□□  □□□□ OmniSS7 □ SS7 □□□□□□□□□□□

短信中心架构



启用 **SMSc** 模式

OmniSS7 可以在不同模式下运行。要将其用作 SMSc，您需要在配置中启用 SMSc 模式。

切换到 **SMSc** 模式

OmniSS7 的 `config/runtime.exs` 包含三个预配置的操作模式。要启用 SMSc 模式：

1. 打开 `config/runtime.exs`

2. 查找三个配置部分（第 53-204 行）：
    - 配置 1：STP 模式（第 53-95 行）
    - 配置 2：HLR 模式（第 97-142 行）
    - 配置 3：SMSc 模式（第 144-204 行）
3. 注释掉任何其他活动配置（在每行前添加 `#`）
4. 取消注释 SMSc 配置（从第 144-204 行删除 `#`）
5. 根据需要自定义配置参数
6. 重启应用程序：`iex -S mix`

SMSc 模式配置

完整的 SMSc 配置如下所示：

```
config :omniss7,
  # 模式标志 - 启用 STP + SMSc 功能
  # 注意: map_client_enabled 为 true, 因为 SMSc 需要路由能力
  map_client_enabled: true,
  hlr_mode_enabled: false,
  smsc_mode_enabled: true,

  # OmniMessage 后端 API 配置
  smsc_api_base_url: "https://10.179.3.219:8443",
  # SMSc 注册时的标识
  smsc_name: "ipsmgw",
  # 短信操作的服务中心 GT 地址
  smsc_service_center_gt_address: "5551234567",

  # 自动刷新配置（后台 SMS 队列处理）
  auto_flush_enabled: true,
  auto_flush_interval: 10_000,
  auto_flush_dest_smsc: "ipsmgw",
  auto_flush_tps: 10,

  # M3UA 连接配置
  # 作为 ASP 连接以发送/接收 MAP SMS 操作
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :stp_client_asp,
    # 本地端点（SMSc 系统）
    local_ip: {10, 179, 4, 12},
    local_port: 2905,
    # 远程 STP 端点
    remote_ip: {10, 179, 4, 10},
    remote_port: 2905,
    routing_context: 1
  }

config :control_panel,
  use_additional_pages: [
    {SS7.Web.EventsLive, "/events", "SS7 事件"},
    {SS7.Web.TestClientLive, "/client", "SS7 客户端"},
    {SS7.Web.M3UAStatusLive, "/m3ua", "M3UA"},
    {SS7.Web.RoutingLive, "/routing", "路由"},
    {SS7.Web.RoutingTestLive, "/routing_test", "路由测试"},
    {SS7.Web.SmscLinksLive, "/smsc_links", "SMSc 链接"}
```

```
  ],
  page_order: ["/events", "/client", "/m3ua", "/routing",
"/routing_test", "/smsc_links", "/application", "/configuration"]
```

自定义的配置参数

有关��有配置参数的完整参考，请参见 配置参考。

□□	□□	□□	□□
`smsc_api_base_url`	□□□□	*□□*	**OmniMessag** □□ API □□
`smsc_name`	□□□□	`"{hostname}_SMSc"`	□□ SMSc □□□□□□ □□
`smsc_service_center_gt_address`	□□□□	*□□*	□□□□□□□□□
`auto_flush_enabled`	□□	`true`	□□□□□□□□□
`auto_flush_interval`	□□	`10_000`	□□□□□□□□□□□
`auto_flush_dest_smsc`	□□□□	*□□*	□□□□□□ SMSC □ □
`auto_flush_tps`	□□	`10`	□□□□□□□□□/□□
`local_ip`	□□	*□□*	□□ SMSc □□□ IP □□
`local_port`	□□	`2905`	□□ SCTP □□
`remote_ip`	□□	*□□*	SS7 □□□ STP IP □□

参数	类型	默认	描述
`remote_port`	整数	`2905`	远程 SCTP 端口
`routing_context`	整数	`1`	M3UA 路由上下文 ID

启用 **SMSc** 模式时发生的事情

当 `smsc_mode_enabled: true` 和 `map_client_enabled: true` 时，Web UI 将显示：

- □ **SS7** 事件 - 事件日志
- □ **SS7** 客户端 - MAP 操作测试
- □ **M3UA** - 连接状态
- □ □□ - 路由表管理（STP 启用）
- □ □□□□ - 路由测试（STP 启用）
- □ **SMSc** 链接 - SMSc API 状态 + SMS 队列管理 ← *SMSc □□*
- □ □□ - 系统监控
- □ □□ - 配置查看器

HLR 链接 选项卡将被隐藏。

重要说明

- SMSc 模式需要 `map_client_enabled: true` 以获得路由能力
- **OmniMessage** 后端：OmniMessage API 后端必须在配置的 `smsc_api_base_url` 可访问
- 前端注册：系统每 **5** 分钟 自动通过 `SMS.FrontendRegistry` 模块注册到 OmniMessage
- **API** 请求超时：所有 OmniMessage API 请求都有 硬编码的 **5** 秒超时
- **MAP** 请求超时：所有 MAP 请求（SRI-for-SM、MT-ForwardSM 等）都有 硬编码的 **10** 秒超时
- 自动刷新会在后台自动处理 SMS 队列
- 需要与 STP 建立 M3UA 连接以发送/接收 MAP SMS 操作
- 更改模式后，必须重启应用程序以使更改生效

- **Web UI**: 有关使用 Web 界面的信息，请参见 Web UI 指南
- **API** 访问：有关 REST API 文档和 Swagger UI 访问的信息，请参见 API 指南

HTTP API 配置

OmniMessage 后端设置

OmniSS7 通过 HTTPS REST API 与 OmniMessage 通信，以管理消息交付、跟踪订阅者状态并注册为活动前端：

```
config :omniss7,
  # OmniMessage API 基础 URL
  smsc_api_base_url: "https://10.5.198.200:8443",
  # SMSC 注册标识符（如果为空，默认为 hostname_SMSc）
  smsc_name: "omni-smsc01",
  # 短信操作的服务中心 GT 地址
  smsc_service_center_gt_address: "5551234567"
```

配置参数：

参数	类型	必需	默认	
`smsc_api_base_url`	字符串	是	`"https://localhost:8443"`	C A
`smsc_name`	字符串	否	`""`（使用 `"{hostname}_SMSc"`）	[S
`smsc_service_center_gt_address`	字符串	否	`"5551234567"`	[[C [F [[S

前端注册

系统在启动时会自动注册自身到 OmniMessage，并且每 **5** 分钟通过 `SMS.FrontendRegistry` 模块重新注册。这使 OmniMessage 能够：

- 跟踪活动端以进行负载均衡
- 监控正常运行时间和健康状态
- 收集配置信息
- 管理多个前端之间的分布式 SMS 路由

实现细节：

- 注册间隔：5 分钟（硬编码）
- 过程：当 `smsc_mode_enabled: true` 时自动启动

注册有效负载：

```
{
  "frontend_name": "omni-smsc01",
  "configuration": "{...}",
  "frontend_type": "SS7",
  "hostname": "smsc-server01",
  "uptime_seconds": 12345
}
```

注意：前端名称取自 `smsc_name` 配置参数。如果未设置，则默认为 `"{hostname}_SMSc"`。

OmniMessage API 通信

当 OmniSS7 从 SS7 网络接收 MAP 操作或处理消息队列时，它与 OmniMessage 通信以：

- 注册为活动前端 并报告健康状态
- 提交从订阅者接收的移动发起 **(MO)** 消息
- 从队列中检索移动终端 **(MT)** 消息以进行交付
- 更新交付状态 以获取成功/失败报告
- 查询路由信息 以进行消息转发

端点	方法	目的	请求体
`/api/frontends`	POST	注册前端实例	`{"frontend_name": "...", "frontend_type": "SMSc", "hostname": "...", "uptime_seconds": ...}`
`/api/messages_raw`	POST	插入新 SMS 消息	`{"source_msisdn": "...", "source_smsc": "...", "message_body": "..."}`
`/api/messages`	GET	获取消息队列	头部: `smsc: <smsc_name>`
`/api/messages/{id}`	PATCH	标记消息为已交付	`{"deliver_time": "...", "dest_smsc": "..."}`
`/api/messages/{id}`	PUT	更新消息状态	`{"dest_smsc": null}`
`/api/locations`	POST	插入/更新订阅者位置	`{"msisdn": "...", "imsi": "...", "location": "...", "ims_capable": true, "csfb": false, "expires": "...", "user_agent": "...", "ran_location": "...", "imei": "...", "registered": "..."}`
`/api/events`	POST	添加事件跟踪	`{"message_id": ..., "name": "...", "description": "..."}`
`/api/status`	GET	健康检查	-

API 响应格式

所有 API 响应均使用 JSON 格式，遵循以下约定:

- 成功响应：HTTP 200-201，JSON 主体包含结果数据
- 错误响应：HTTP 4xx/5xx，响应主体中包含错误详细信息
- 时间戳：ISO 8601 格式（例如，`"2025-10-21T12:34:56Z"`）
- 消息 **ID**：整数或字符串标识符

API 客户端模块

SMS 系统由三个主要模块组成：

1. SMSc.APIClient

主要 API 客户端模块，提供与 OmniMessage 的所有 HTTP API 通信：

- `frontend_register/4` - 向 OmniMessage 注册前端
- `insert_message/3` - 插入原始 SMS 消息（与 Python 兼容的 3 参数版本）
- `insert_location/9` - 插入/更新订阅者位置数据
- `get_message_queue/2` - 从队列中检索待处理消息
- `mark_dest_smsc/3` - 标记消息为已交付或失败
- `add_event/3` - 为消息添加事件跟踪
- `flush_queue/2` - 处理待处理消息（SRI-for-SM + MT-forwardSM）
- `auto_flush/2` - 持续的队列处理循环

2. SMS.FrontendRegistry

处理与后端的定期前端注册：

- 启动时自动注册
- 每 5 分钟重新注册
- 使用配置中的 `smsc_name`（回退到主机名）
- 收集系统配置和正常运行时间信息

3. SMS.Utils

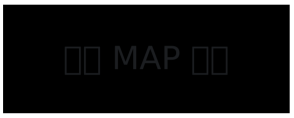
用于 SMS 操作的实用函数：

- `generate_tp_scts/0` - 生成 TPDU 格式的 SMS 时间戳

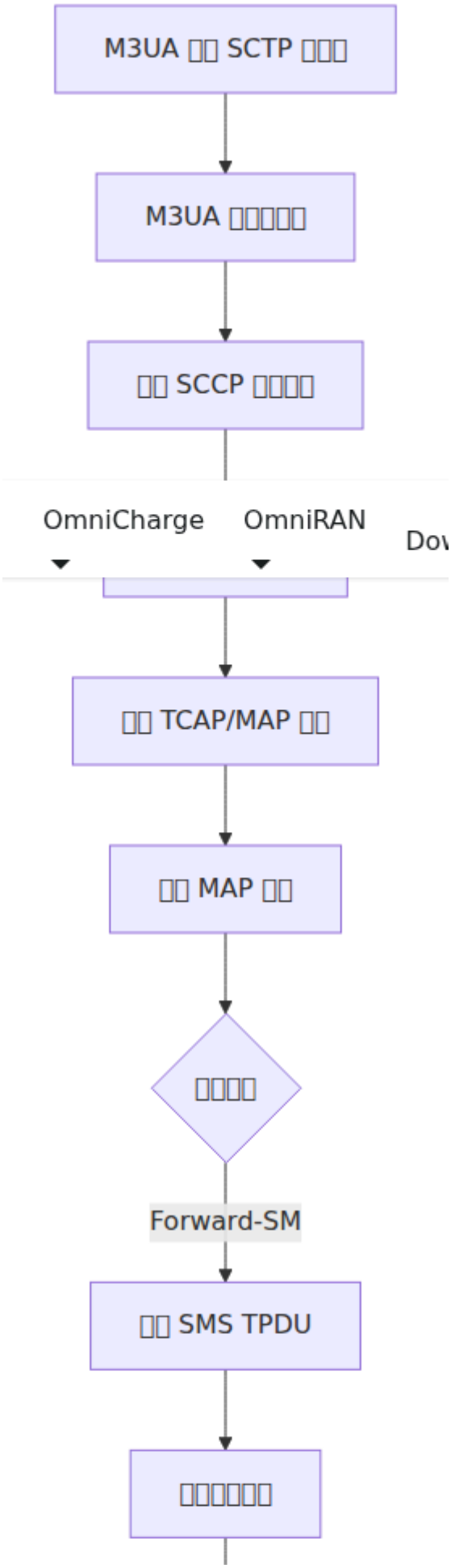
短信消息流

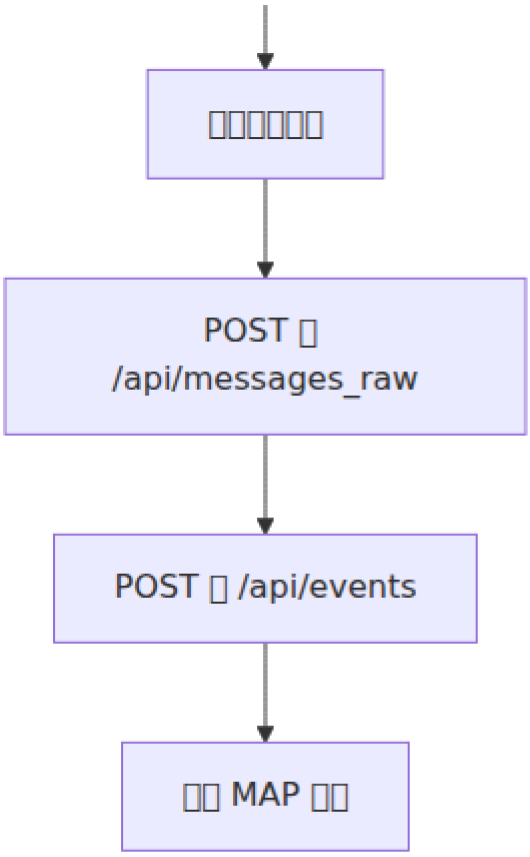
入站 **SMS** 流（移动发起）


M3UA □□ SCTP □□□
M3UA □□□□□
□□ SCCP □□□□
□□ SCCP □□
□□ TCAP/MAP □□
□□ MAP □□
□□□□
Forward-SM
□□ SMS TPDU
□□□□□□□



出站 **SMS** 流（移动终端）

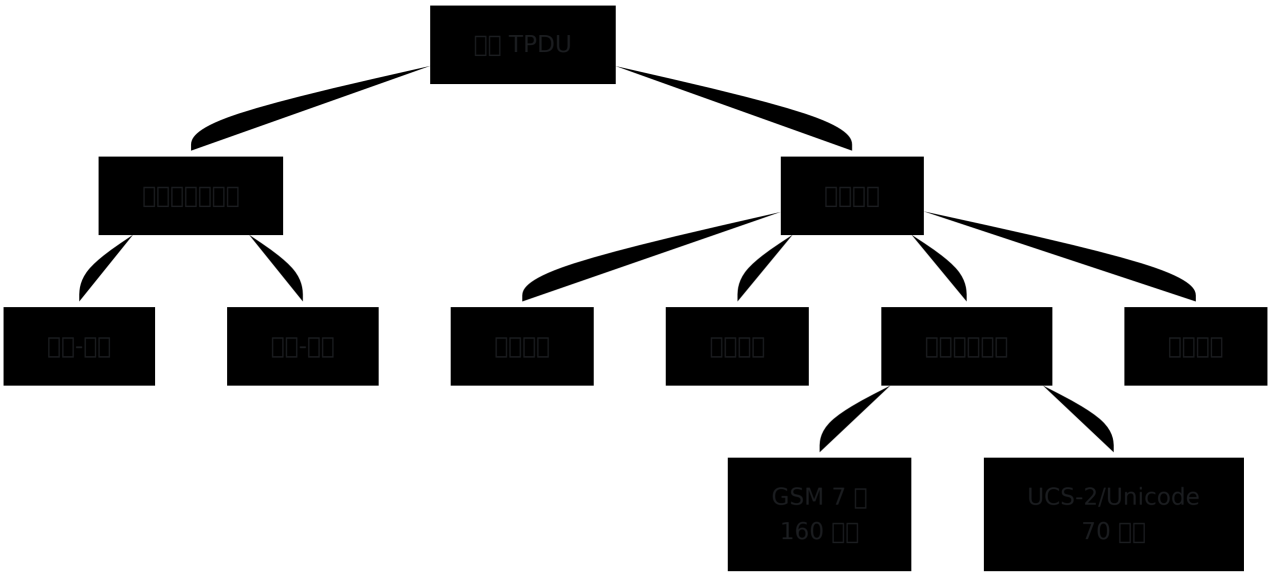

M3UA □□ SCTP □□□
M3UA □□□□□
□□ SCCP □□□□
OmniCharge
OmniRAN
Dov
□□ TCAP/MAP □□
□□ MAP □□
□□□□
Forward-SM
□□ SMS TPDU
□□□□□□



关键步骤解释：

- **SRI-for-SM** 请求：SMSc 向 HLR 查询目标 MSISDN，以确定将 SMS 消息路由到哪里。HLR 响应：
  - 一个合成 IMSI（从 MSISDN 计算得出以保护隐私） - 参见 MSISDN ↔ IMSI 映射
  - SMSC GT 地址（网络节点编号），MT-ForwardSM 应该发送到这里
  - 有关此工作原理的完整细节，请参见 HLR 指南中的 SRI-for-SM
- **MT-forwardSM** 请求：一旦获得路由信息，SMSc 将实际的 SMS 消息发送到服务该订阅者的 MSC/VLR

SMS TPDU 结构



警报服务中心处理

SMSc 可以接收来自 HLR 的 **alertServiceCenter** 消息，以跟踪订阅者的可达性状态。

有关 HLR 如何发送 alertServiceCenter 消息的信息，请参见 HLR 指南中的警报服务中心集成。

什么是 **alertServiceCenter**?

当订阅者在 HLR 执行 UpdateLocation（即，注册到新的 VLR/MSC）时，HLR 可以通过发送 **alertServiceCenter**（MAP 操作码 64）消息通知 SMSc 系统，订阅者现在可达。

配置

位置过期时间在 HLR 中配置:

```
config :omniss7,
  # SMSc 接收 alertServiceCenter 时的位置过期时间（默认: 48 小时）
  hlr_alert_location_expiry_seconds: 172800
```

行为

当 SMSc 接收到 alertServiceCenter 消息时：

1. 解码 **MSISDN**：从消息中提取订阅者的 MSISDN（TBCD 格式）
2. 去除 **TON/NPI** 前缀：去除常见前缀，如 "19"、"11"、"91"（例如，"19123123213" → "123123213"）
3. 计算 **IMSI**：使用与 SRI-for-SM 相同的映射生成合成 IMSI
4. **POST** 到 **/api/location**：使用以下内容更新位置数据库：
   - `msisdn`：订阅者的电话号码（已清理）
   - `imsi`：合成 IMSI
   - `location`：SMSc 名称（例如，"ipsmgw"）
   - `expires`：当前时间 + `hlr_alert_location_expiry_seconds`
   - `csfb`：true（订阅者可通过电路交换回退到达）
   - `ims_capable`：false（这是 2G/3G CS 注册，而不是 IMS/VoLTE）
   - `user_agent`：发送警报的 HLR GT（用于跟踪）
   - `ran_location`："SS7"
5. 在 **SMSc** 订阅者跟踪器中跟踪：记录订阅者与 HLR GT，状态=活动，消息计数器为 0
6. 发送 **ACK**：回复 HLR 以确认 alertServiceCenter

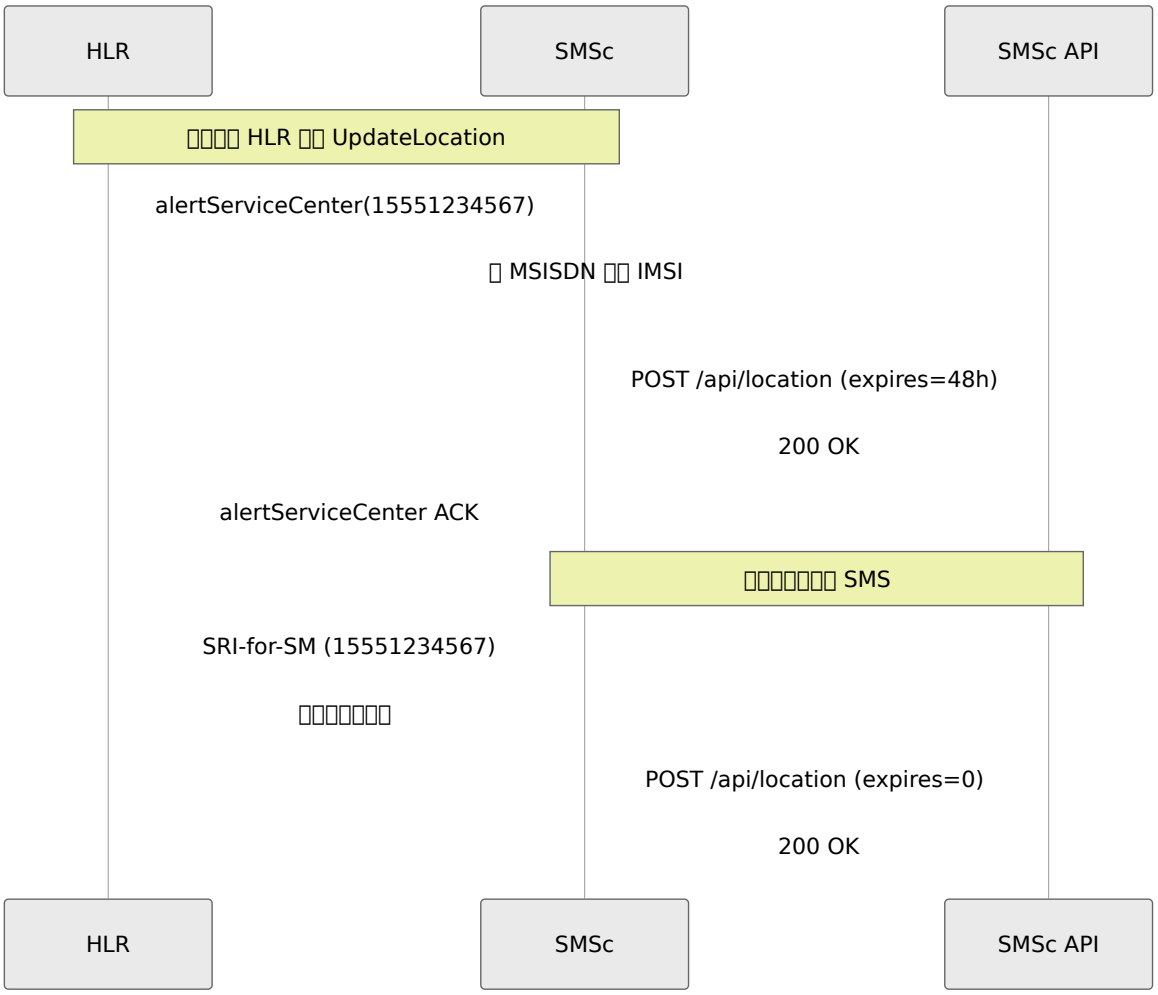
缺席订阅者处理

当 SMSc 尝试交付消息并在 SRI-for-SM 中收到 "缺席订阅者" 错误时（有关 SRI-for-SM 的更多信息，请参见 HLR 指南中的 SRI-for-SM）：

1. 检测缺席：SRI-for-SM 回 `absentSubscriberDiagnosticSM` 错误
2. 过期位置：POST 到 /api/location，`expires=0` 以标记订阅者为不可达
3. 用户代理：设为 "SS7_AbsentSubscriber" 以识别来源
4. 更新跟踪器：在 SMSc 订阅者跟踪器中将订阅者标记为 `failed`

这确保位置数据库和跟踪器准确反映订阅者的可达性状态。

□□□



API □□

POST /api/location

```json
{
  "msisdn": "15551234567",
  "imsi": "001010123456789",
  "location": "ipsmgw",
  "ims_capable": false,
  "csfb": true,
  "expires": "2025-11-01T12:00:00Z",
  "user_agent": "15551111111",
  "ran_location": "SS7",
  "imei": "",
  "registered": "2025-10-30T12:00:00Z"
}
```

注意：`user_agent` 字段包含发送 alertServiceCenter 的 HLR GT，使 SMSc 能够跟踪提供位置更新的 HLR。

对于缺席的订阅者，`expires` 设置为当前时间（立即过期）。

循环预防

SMSc 实施自动循环预防，以避免在消息源自 SS7 网络时出现无限消息路由循环。

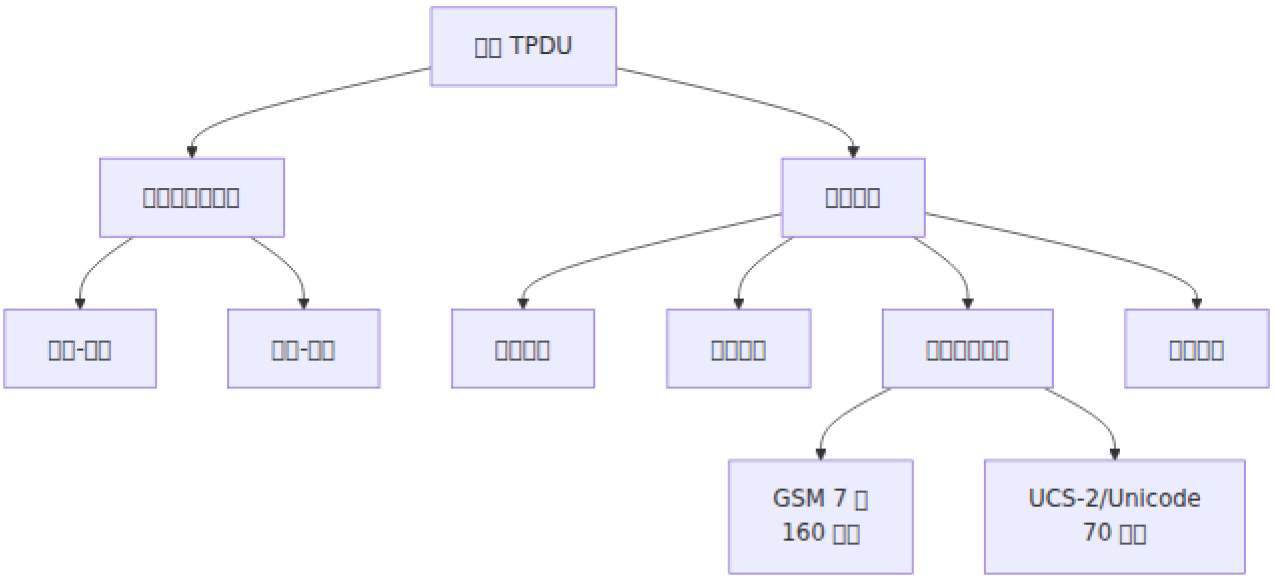
为什么循环预防很重要

当 SMSc 从 SS7 网络接收移动发起 (MO) 短信时，它将其插入消息队列，并带有 `source_smsc` 字段以标识其来源（例如，`"SS7_GT_15551234567"`）。如果没有循环预防，这些消息可能会：

1. 从 SS7 网络接收 → 使用 `source_smsc` 排队，包含 "SS7"
2. 从队列中检索 → 处理以进行交付
3. 发送回 SS7 网络 → 创建循环

它是如何工作的

SMSc 在消息处理期间自动检测并防止循环：



实现

在处理来自队列的消息时，SMSc 检查 `source_smsc` 字段：

- 如果 **`source_smsc`** 包含 **"SS7"**：
  - 跳过消息
  - 添加事件：`"循环预防"`，描述跳过原因
  - 通过 PUT 请求将消息标记为失败
  - 以警告级别记录
- 否则：
  - 正常处理消息
  - 继续 SRI-for-SM 和 MT-ForwardSM 操作

源 **SMSC** 值

消息可以具有各种 `source_smsc` 值：

来源	示例值	操作
SS7 网络 (MO-FSM)	`"SS7_GT_15551234567"`	跳过 - 循环预防
外部 API/SMPP	`"ipsmgw"` 或 `"api_gateway"`	正常处理
其他 SMSc	`"smsc-node-01"`	正常处理

事件跟踪

当由于循环预防而跳过消息时，会记录一个事件：

```
{
  "message_id": 12345,
  "name": "循环预防",
  "description": "消息跳过 - source_smsc 'SS7_GT_15551234567' 包含
'SS7'，防止消息循环"
}
```

该事件在以下位置可见：

- **Web UI**: SS7 事件页面 (`/events`)
- 数据库：通过 API 的 `events` 表
- 日志：警告级别日志条目

配置

循环预防始终 启用，无法禁用。这是防止消息循环导致网络中断的关键安全功能。

示例场景

场景：移动订阅者通过 SS7 网络发送 SMS

1. 移动电话 → MSC/VLR → SMSc (通过 MO-ForwardSM)
2. SMSc 从 GT 15551234567 接收 MO-FSM
3. SMSc 插入队列: source_smsc = "SS7_GT_15551234567"
4. 自动刷新从队列中检索消息
5. SMSc 检测到 source_smsc 中的 "SS7" → 跳过
6. 记录事件: "循环预防"
7. 消息标记为失败
8. 不发送 SRI-for-SM 或 MT-ForwardSM (防止循环)

如果没有循环预防，第 8 步将把消息发送回 SS7 网络，可能会创建无限循环。

SMSc 订阅者跟踪

SMSc 包含一个 订阅者跟踪器 GenServer，实时维护基于 alertServiceCenter 消息和消息交付尝试的订阅者状态。

目的

跟踪器提供:

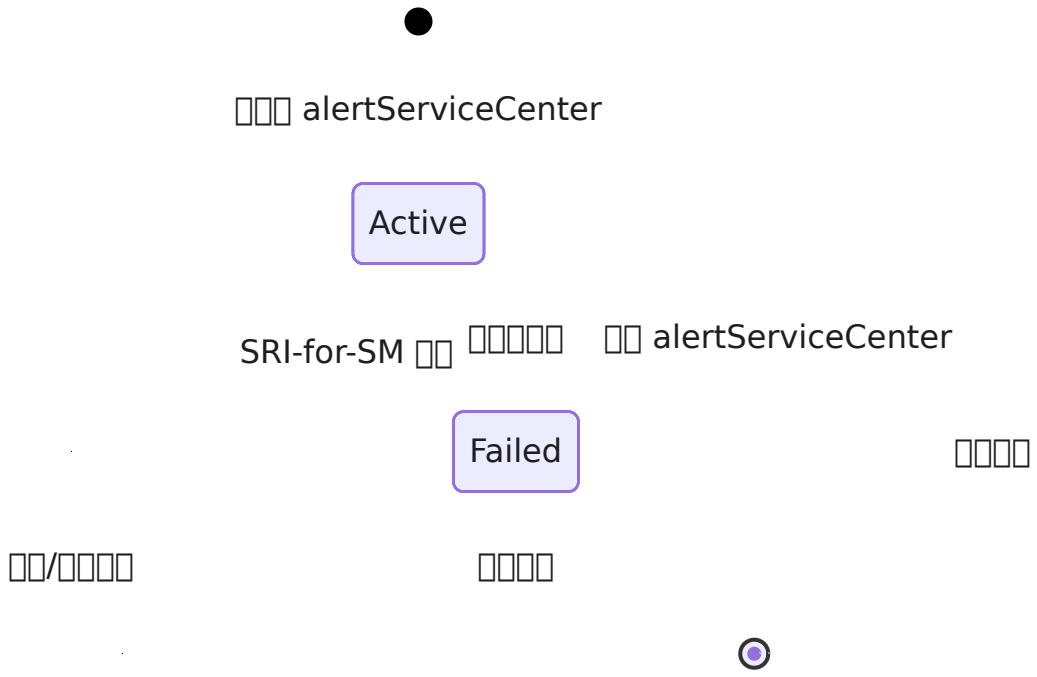
- 可达性监控: 当前可达的订阅者
- **HLR** 跟踪: 发送 alertServiceCenter 的 HLR
- 消息计数器: 每个订阅者发送/接收的消息数量
- 失败跟踪: 当交付尝试失败时将订阅者标记为失败
- **Web UI** 可见性: 实时仪表板显示所有跟踪的订阅者

跟踪的信息

对于每个订阅者，跟踪器存储:

□□	□□	□□
`msisdn`	□□□□□□□□□□□□	"15551234567"
`imsi`	□□□□ IMSI	"001010123456789"
`hlr_gt`	□□ alertServiceCenter □ HLR GT	"15551111111"
`messages_sent`	□□□ MT-FSM □□□□	5
`messages_received`	□□□ MO-FSM □□□□	2
`status`	`:active` □ `:failed`	`:active`
`updated_at`	□□□□□□□ Unix □□□	1730246400

□□□□



□□

□□□□ alertServiceCenter □□

- 创建或更新订阅者条目
- 设置 `status = :active`
- 记录 HLR GT
- 重置或保留消息计数器

当 **SRI-for-SM** 成功时：

- 增加 `messages_sent` 计数器
- 更新 `updated_at` 时间戳

当 **SRI-for-SM** 失败时：

- 设置 `status = :failed`
- 保留在跟踪器中以进行监控

当订阅者被移除时：

- 从 ETS 表中删除
- 不再出现在 Web UI 中

Web UI - SMSc 订阅者页面

路径：`/smsc_subscribers`
自动刷新：每 2 秒

注意：此页面仅在 SMSc 模式下运行时可用。在 `config/runtime.exs` 中取消注释 SMSc 配置后，必须重启应用程序以使路由可用。

SMSc 订阅者 页面提供所有跟踪订阅者的实时监控：

特性

1. 订阅者表
    - MSISDN、IMSI、HLR GT
    - 发送/接收消息计数器
    - 状态徽章（活动/失败）及颜色编码
    - 最后更新时间戳和持续时间

- □□□□□□□□□□□□

2. □□□□

- □□□□□□□
- □□□□□□□□
- □□□□□□□□
- □□ HLR □□□

3. □□

- □□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□

□□□□

```
┌──────────────────────────────────────────────────────────────────────────┐
│ SMSc □□□□□□□                              □□: 3                  │
├──────────────────────────────────────────────────────────────────────────┤
│ MSISDN          IMSI              HLR GT         Msgs  □□        │
│                                                   S/R                    │
├──────────────────────────────────────────────────────────────────────────┤
│ 15551234567     001010123456789   15551111111    5/2   ● □□      │
│ 15559876543     001010987654321   15551111111    0/0   ● □□      │
│ 15551112222     001010111222233   15552222222    3/1   ○ □□      │
└──────────────────────────────────────────────────────────────────────────┘
      ???

□□□  □□: 3  |  □□: 2  |  □□: 1  |  □□ HLR: 2
```

API □□

□□□□□□□□□□□□□□□□□□□□

```
# □□□□ alertServiceCenter □□□
SMSc.SubscriberTracker.alert_received(msisdn, imsi, hlr_gt)

# □□□□□□□□
SMSc.SubscriberTracker.message_sent(msisdn)
SMSc.SubscriberTracker.message_received(msisdn)

# □□□□□□SRI-for-SM □□□
SMSc.SubscriberTracker.mark_failed(msisdn)

# □□□□□□
SMSc.SubscriberTracker.remove_subscriber(msisdn)

# □□□□
SMSc.SubscriberTracker.get_active_subscribers()
SMSc.SubscriberTracker.get_subscriber(msisdn)
SMSc.SubscriberTracker.count_subscribers()
SMSc.SubscriberTracker.clear_all()
```

□□

□□□□□□□□□□□□□□□

- **alertServiceCenter** □□□□□□□□□□□□□□□□ `alert_received/3`
- **SRI-for-SM** □□□□□□□□□□□□□□ `messages_sent`
- □□□□□□□□□□□□□□□□□□□□□ `mark_failed/1`
- □□□□□□□□□ SRI-for-SM □□□□□ `mark_failed/1`

□□□□ SMS □□

□��□□ □□□□□□□□□□ SMS □□□

□□□□□□□□□□□□ □□□□□□□□□□□□□□□□

配置

```
config :omniss7,
  auto_flush_enabled: true,              # 启用/禁用自动刷新
  auto_flush_interval: 10_000,           # 轮询间隔（毫秒）
  auto_flush_dest_smsc: nil,             # 过滤器：nil = 所有
  auto_flush_tps: 10                     # 每秒最大事务数
```

工作原理

1. 轮询：每 `auto_flush_interval` 毫秒，查询 API 以获取待处理消息
2. 过滤：可选地通过 `auto_flush_dest_smsc` 进行过滤
3. 速率限制：每个周期最多处理 `auto_flush_tps` 消息
4. 交付：对于每条消息：
   - 发送 **SRI-for-SM**（发送短消息路由信息）到 HLR 以获取路由信息
     - HLR 回一个合成 IMSI，从 MSISDN 计算得出
     - HLR 回 SMSC GT 地址，MT-ForwardSM 应该发送到这里
     - 有关完整文档，请参见 HLR 指南中的 SRI-for-SM 详细信息。
   - 成功时，发送 **MT-forwardSM** 到 MSC/VLR
   - 通过 API 更新消息状态（已交付/失败）
   - 通过 API 添加事件跟踪

> 技术深入：有关 SRI-for-SM 工作原理的完整解释，包括 MSISDN 到 IMSI 映射、服务中心 GT 地址配置和隐私保护的合成 IMSI 生成，请参见 HLR 配置指南中的 SRI-for-SM 部分。

SMSc 标

可用标

SMS 队列标：

- `smsc_queue_depth` - 当前待处理消息数量
- `smsc_messages_delivered_total` - 成功交付的总消息数
- `smsc_messages_failed_total` - 交付失败的总消息数

- `smsc_delivery_duration_milliseconds` - 交付时间的直方图

示例查询：

```
# 当前队列深度
smsc_queue_depth

# 交付成功率（过去 5 分钟）
rate(smsc_messages_delivered_total[5m]) /
(rate(smsc_messages_delivered_total[5m]) +
rate(smsc_messages_failed_total[5m]))

# 平均交付时间
rate(smsc_delivery_duration_milliseconds_sum[5m]) /
rate(smsc_delivery_duration_milliseconds_count[5m])
```

故障排除 SMSc

问题：消息未交付

检查：

1. 验证自动刷新是否启用
2. 检查数据库连接
3. 监控日志以查找错误
4. 验证 M3UA 连接是否处于活动状态
5. 检查 TPS 限制

问题：队列深度过高

可能原因：

- TPS 限制过低
- HLR 超时问题
- 网络连接问题

- 无效的目的号码

解决方案：

- 增加 `auto_flush_tps`
- 检查 HLR 可用性
- 审查失败的消息日志

MT-forwardSM API

通过 API 发送 SMS

API 端点：`POST /api/MT-forwardSM`

请求：

```
{
  "imsi": "234509876543210",
  "destination_serviceCentre": "447999555111",
  "originating_serviceCenter": "447999123456",
  "smsPDU":
"040B917477218345F60000157030185714 0C0BD4F29C0E9281C4E1F11A"
}
```

响应：

```
{
  "result": "success",
  "message_id": "12345"
}
```

相关文档

OmniSS7 文档:

- ← 回主文档
- HLR 配置指南 - HLR 模式设置和操作
  - **SRI-for-SM** 技术细节 - 有关 MSISDN 到 IMSI 映射和服务中心配置的完整文档
- 常见功能指南 - Web UI、API、监控
- MAP 客户端指南 - MAP 操作
- 技术参考 - 协议规范

OmniMessage 文档：有关消息路由配置、队列管理、交付跟踪、速率限制和分析的信息，请参考 **OmniMessage** 产品文档。OmniMessage 包含所有消息路由逻辑、队列重试算法、交付报告处理和业务规则引擎。

OmniSS7 由 Omnitouch 网络服务提供

M3UA STP 配置□南

← □回主文档

本指南提供了使用 OmniSS7 作为 信令传输点 **(STP)** 的详细配置。

目录

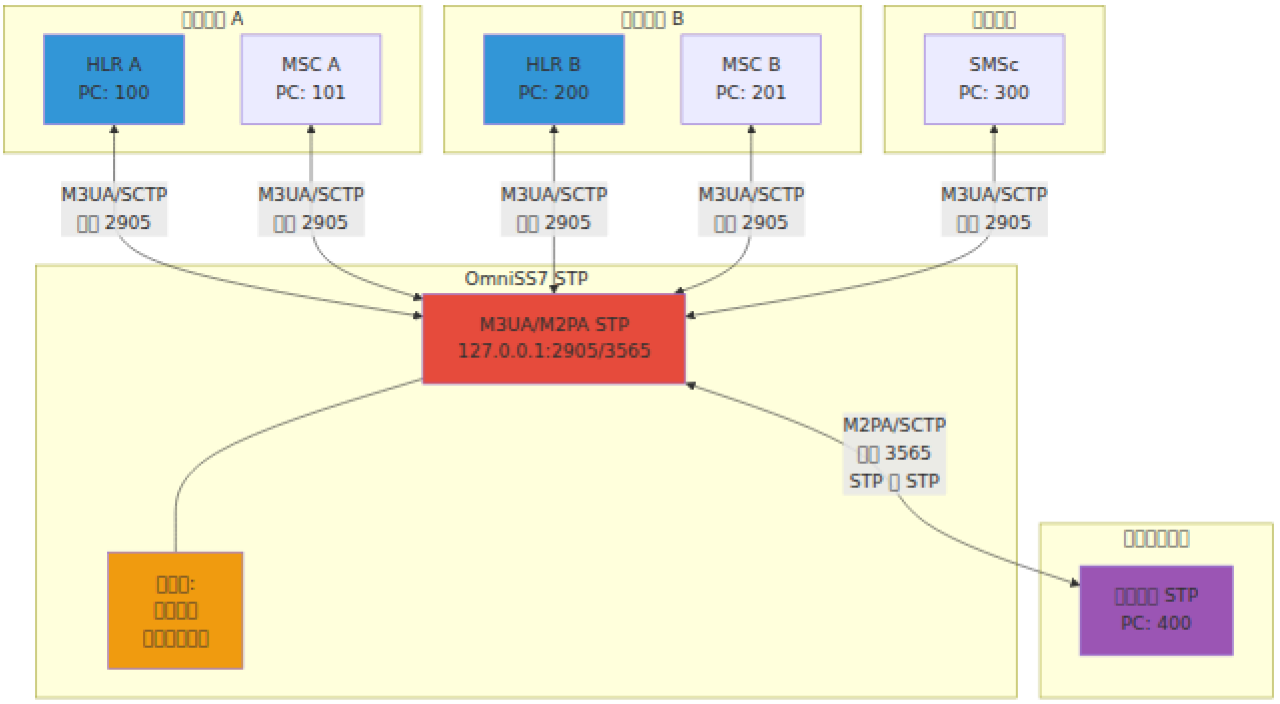
什么是信令传输点 **(STP)**?

信令传输点 **(STP)** 是 SS7 和基于 IP 的信令网络中的关键网络元素，负责在网络节点之间路由信令消息。

STP 功能

- 消息路由：根据目标点码（PC）或全局标题（GT）路由 SS7 信令流量
- 协议转换：将传统 SS7 网络与基于 IP 的 M3UA/SCTP 网络连接起来
- 负载分配：使用基于优先级的路由将流量分配到多个目标
- 网络网关：连接不同的信令网络和服务提供商
- 拓扑隐藏：可以重写地址以隐藏内部网络拓扑

STP 网络图



STP 网络角色解释

ASP（应用服务器进程）

- 角色：连接到远程 SGP/STP 的客户端
- 方向：出站连接
- 用例：您的 STP 连接到合作伙伴网络的 STP

SGP（信令网关进程）

- 角色：接受来自 ASP 的连接的服务器
- 方向：入站连接
- 用例：合作伙伴网络连接到您的 STP

AS（应用服务器）

- 定义：一个或多个 ASP 的逻辑分组

- 目的：提供冗余和负载共享
- 用例：多个 ASP 为同一目标服务

启用 M3UA STP 模式

OmniSS7 可以在不同模式下运行。要将其用作 STP，您需要在配置中启用 STP 模式。

切换到 STP 模式

OmniSS7 的 `config/runtime.exs` 包含三个预配置的操作模式。要启用 STP 模式：

1. 打开 `config/runtime.exs`
2. 查找三个配置部分（第 53-174 行）：
    - 配置 1：STP 模式（第 53-85 行）
    - 配置 2：HLR 模式（第 87-123 行）
    - 配置 3：SMSc 模式（第 125-174 行）
3. 注释掉当前活动配置（在每行前添加 `#`）
4. 取消注释 STP 配置（从第 53-85 行移除 `#`）
5. 根据需要自定义配置参数
6. 重启应用程序：`iex -S mix`

STP 模式配置

完整的 STP 配置如下所示：

```
config :omniss7,
  # 模式标志 - 仅启用 STP 功能
  map_client_enabled: true,
  hlr_mode_enabled: false,
  smsc_mode_enabled: false,

  # M3UA 连接配置
  # 作为 ASP（应用服务器进程）连接到远程 STP/SGW
  map_client_m3ua: %{
    mode: "ASP",
    callback: {MapClient, :handle_payload, []},
    process_name: :stp_client_asp,
    # 本地端点（此系统）
    local_ip: {10, 179, 4, 10},
    local_port: 2905,
    # 远程 STP/SGW 端点
    remote_ip: {10, 179, 4, 11},
    remote_port: 2905,
    routing_context: 1
  }
```

自定义的配置参数

有关所有配置参数的完整参考，请参见 配置参考。

参数	类型	默认值	描述	示例
`map_client_enabled`	布尔值	`true`	启用 MAP 客户端和路由功能	`true`
`local_ip`	元组或列表	*□□*	您系统的 IP 地址。单个：`{10, 0, 0, 1}` 或多重：`[{10, 0, 0, 1}, {10, 0, 0, 2}]`	`{10, 179, 4, 10}`
`local_port`	整数	`2905`	本地 SCTP 端口	`2905`
`remote_ip`	元组或列表	*□□*	远程 STP/SGW IP 地址。单个或多重	`{10, 179, 4, 11}`
`remote_port`	整数	`2905`	远程 SCTP 端口	`2905`
`routing_context`	整数	`1`	M3UA 路由上下文 ID	`1`
`enable_gt_routing`	布尔值	`false`	启用全局标题路由（除了 PC 路由）	`true`

> 提示：通过为 `local_ip` 和/或 `remote_ip` 提供 IP 地址列表来使用 SCTP 多宿主，以启用自动故障转移。请参见 SCTP 多宿主指南。

启用 **STP** 模式后发生的情况

当 `map_client_enabled: true` 时，Web UI 将显示：

- □ **SS7 事件** - 事件日志
- □ **SS7 客户端** - MAP 操作测试
- □ **M3UA** - 连接状态
- □ **□□** - 路由表管理 ← *STP □□*
- □ **□□□□** - 路由测试 ← *STP □□*
- □ **□□** - 系统监控
- □ **□□** - 配置查看器

HLR 链接 和 **SMSc** 链接 选项卡将被隐藏。

重要注意事项

- SCTP 协议（IP 协议 132）必须允许通过防火墙
- 默认 M3UA 端口为 2905（行业标准）
- 确保有足够的系统资源来处理路由流量
- 路由持久性：通过 Web UI 或 API 配置的所有路由都存储在 **Mnesia** 数据库 中，并且 在重启后仍然有效
- 配置合并：`runtime.exs` 中的路由在启动时加载，并与 Mnesia 路由合并
- 更改模式后，必须重启应用程序以使更改生效
- **Web UI**：有关通过 Web 界面管理路由的信息，请参见 Web UI 指南
- **API** 访问：有关 REST API 文档和 Swagger UI 访问的信息，请参见 API 指南

独立 STP 模式

除了在 `map_client_enabled: true` 时可用的 STP 路由功能外，您还可以运行一个 独立的 **M3UA STP** 服务器，该服务器监听传入连接。

启用独立 STP

将以下配置添加到 `config/runtime.exs`：

```
config :omniss7,
  m3ua_stp: %{
    enabled: true,
    local_ip: {127, 0, 0, 1},     # 监听的 IP 地址
    local_port: 2905,              # 监听的端口
    point_code: 100                # 此 STP 自身的点码
  }
```

STP 配置参数

参数	类型	默认值	描述	示例
`enabled`	布尔值	`false`	启用独立 STP 服务器	`true`
`local_ip`	元组	`{127, 0, 0, 1}`	监听连接的 IP 地址	`{0, 0, 0, 0}`
`local_port`	整数	`2905`	监听的端口	`2905`
`point_code`	整数	*□□*	此 STP 自身的 SS7 点码	`100`

何时使用独立 **STP**

- 纯路由：当您只需要 M3UA 路由而不需要 MAP 客户端功能时
- 中央 **STP**：为多个网络元素创建一个中央信令路由器
- 中心架构：通过中央 STP 连接多个 HLR、MSC 和 SMSC

注意：如果您需要出站连接和入站 STP 功能，可以同时启用 `map_client_m3ua` 和 `m3ua_stp`。

路由表持久性（**Mnesia**）

所有路由表（对等体、点码路由和全局标题路由）都存储在 **Mnesia** 数据库中以实现持久性。

路由工作原理

1. **Runtime.exs** 路由：在应用程序启动时加载 `config/runtime.exs` 中定义的路由，位于 `m3ua_peers`、`m3ua_routes` 和 `m3ua_gt_routes` 下
2. **Web UI** 路由：通过 Web UI 路由页面添加的路由存储在 Mnesia 中
3. 路由合并：在重启时，runtime.exs 路由与现有 Mnesia 路由合并（无重复）
4. 持久性：通过 Web UI 配置的所有路由在应用程序重启后仍然有效

Mnesia □□□□

□□□

□□□□□□□

```
config :omniss7,
  m3ua_peers: [
    # □□□□□□ STP □□□□□□□□□□□□:client□
    %{
      peer_id: 1,                              # □□□□□□
      name: "Partner_STP_West",                # □□□□□□
      role: :client,                           # :client □□□□□□:server
□□□□□
      local_ip: {10, 0, 0, 1},                # □□ IP □□
      local_port: 0,                           # 0 = □□□□□□□
      remote_ip: {10, 0, 0, 10},              # □□□□□□ IP
      remote_port: 2905,                       # □□□□□□□□
      routing_context: 1,                      # M3UA □□□□□□
      point_code: 100,                         # □□□□□□□□
      network_indicator: :international     # :international □
:national
    },

    # □□□□ HLR □□□□□□□□:client□
    %{
      peer_id: 2,
      name: "Local_HLR",
      role: :client,
      local_ip: {10, 0, 0, 1},
      local_port: 0,
      remote_ip: {10, 0, 0, 20},
      remote_port: 2905,
      routing_context: 2,
      point_code: 200,
      network_indicator: :international
    },

    # □□□□□ MSC □□□□□□□□□□□:server□
    # □□ :server □□□STP □□□□□□□
    %{
      peer_id: 3,
      name: "Remote_MSC",
      role: :server,                           # □□□□□□□
      remote_ip: {10, 0, 0, 30},              # □□□□ IP
      remote_port: 2905,                       # □□□□□□□0 = □□□□□□□□□□
      routing_context: 3,
```

```
      point_code: 300,
      network_indicator: :international
    },

    # 动态源端口的入站连接（无端口过滤）
    %{
      peer_id: 4,
      name: "Dynamic_Client",
      role: :server,
      remote_ip: {10, 0, 0, 40},            # 预期源 IP
      remote_port: 0,                        # 0 = 接受来自任何源端口的连接
      routing_context: 4,
      point_code: 400,
      network_indicator: :international
    }
  ]
```

对等体配置参数

参数	类型	必需	描述
`peer_id`	整数	是	对等体的唯一数字标识符
`name`	字符串	是	日志和监控的可读名称
`role`	原子	是	`:client`（出站）或 `:server`（入站）
`local_ip`	元组或列表	是（客户端）	本地 IP 地址。单个：`{10, 0, 0, 1}` 或多个用于 SCTP 多宿主：`[{10, 0, 0, 1}, {10, 0, 0, 2}]`
`local_port`	整数	是（客户端）	本地端口（0 为动态）
`remote_ip`	元组或列表	是	远程对等体 IP 地址。单个：`{10, 0, 0, 10}` 或多个：`[{10, 0, 0, 10}, {10, 0, 0, 11}]`
`remote_port`	整数	是	远程对等体端口（0 为入站 = 接受来自任何源端口）
`routing_context`	整数	是	M3UA 路由上下文标识符
`point_code`	整数	是	此对等体的 SS7 点码
`network_indicator`	原子	否	`:international` 或 `:national`

> **SCTP** 多宿主：为了网络冗余，您可以为 `local_ip` 和 `remote_ip` 配置多个 IP 地址。这使得在一个网络路径失败时能够自动故障转移。请参见 SCTP 多宿主指南 以获取详细的配置示例和最佳实践。

入站连接的源端口过滤

对于 入站连接（角色：`:server`），`remote_port` 参数控制源端口过滤：

- 特定端口（例如，`remote_port: 2905`）：仅接受来自该确切源端口的连接
    - 通过验证源端口提供额外的安全性
    - 当远程对等体使用固定源端口时使用
- 任何端口（`remote_port: 0`）：接受来自任何源端口的连接
    - 当远程对等体使用动态/临时源端口时很有用
    - 仅验证源 IP 地址
    - 更灵活但稍微不安全

示例：

```
# 仅接受来自 10.5.198.200:2905（特定端口）
%{
  peer_id: 1,
  name: "Strict_Peer",
  role: :server,
  remote_ip: {10, 5, 198, 200},
  remote_port: 2905,
  # ... 其他配置
}

# 接受来自 10.5.198.200 的任何源端口
%{
  peer_id: 2,
  name: "Flexible_Peer",
  role: :server,
  remote_ip: {10, 5, 198, 200},
  remote_port: 0,  # 接受来自任何源端口
  # ... 其他配置
}
```

M2PA 协议支持

OmniSS7 支持 **M3UA** 和 **M2PA** 协议用于 SS7 信令传输。

什么是 **M2PA**?

M2PA（MTP2 用户对等适配层）是 IETF 标准化的协议（RFC 4165），用于通过 IP 网络使用 SCTP 传输 SS7 MTP3 消息。

M3UA 与 **M2PA**：关键区别

特性	**M3UA**	**M2PA**
架构	客户端/服务器 (ASP/SGW)	对等
用例	SS7 与 IP 之间的网关	直接点对点链接
链路状态管理	应用层（ASPUP/ASPAC）	MTP2 风格（对齐、证明、就绪）
序列号	无固有排序	24 位 BSN/FSN 用于有序交付
典型部署	SS7 到 IP 网关，STP	节点之间的直接信令链接
RFC	RFC 4666	RFC 4165

协议选择□导

建议：默认使用 **M3UA**。仅在特定需要时使用 **M2PA**。

何时使用 **M3UA**（推荐）

M3UA 是大多数部署的推荐协议：

- **STP** 部署：标准信令传���点实现
- 网关功能：将 SS7 网络与基于 IP 的信令连接
- 网络元素连接：将 HLR、MSC、SMSC 和其他网络元素连接到您的 STP
- 信令网关 **(SGW)**：接受来自多个应用服务器的连接的中央网关
- 灵活拓扑：具有集中控制的客户端/服务器架构
- 多厂商网络：广泛支持的行业标准（RFC 4666）

□□ **M3UA** □□□□□□□**HLR□MSC□SMSC□VLR** □□□□□□□□ **STP□**

□□□□ M2PA□□□□□□□□

M2PA □□□□□□□□□□□□

- **STP □ STP** □□□□□□ STP □□□□□□□□□□□□□□□□□□□□□□□□
- □□ **TDM** □□□□□□□□□□□□□□ M2PA □□□□□□□ SS7 TDM □□
- □□ **MTP2** □□□□□□□□□□□□□□ MTP2 □□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□ M2PA □□□

□□□□□□□□ M2PA □□□□□□□HLR□MSC□SMSC□□□□□□□ STP - □□□ M3UA□M2PA □□□□□ STP □ STP □□□□□□□□□□□□□□□□□□□□□□

□□ M2PA □□□

M2PA □□□□□□□□ M3UA □□□□□□□□□□□□□ `protocol` □□□

M2PA □□□□□

□ M2PA □□□□□□□ `config/runtime.exs` □□ `m3ua_peers` ��□□□□□□□□□□□□□□□□□□□

参数	值	描述
`protocol`	`:m2pa`	指定 M2PA 协议（如果省略，则默认为 `:m3ua`）
`role`	`:client` 或 `:server`	连接方向
`local_port`	整数	本地 SCTP 端口（标准 M2PA 端口为 **3565**）
`remote_port`	整数	远程 SCTP 端口（标准 M2PA 端口为 **3565**）
`point_code`	整数	您的点码
`adjacent_point_code`	整数	远程对等体的点码（M2PA 特定）

注意：M2PA 使用端口 **3565** 作为行业标准（与 M3UA 的端口 2905 不同）。

M2PA 链路状态

M2PA 链接在初始化期间经历几个状态：

1. 关闭 - 未建立连接
2. 对齐 - 初始同步阶段（约 1 秒）
3. 证明 - 链路质量验证（约 2 秒）
4. 就绪 - 链路活动并准备传输流量

链路状态进展确保在交换流量之前进行可靠的信令。

通过 **Web UI** 管理 **M2PA** 对等体

Web UI 的路由页面提供对 M2PA 对等体的完整管理支持：

1. 导航 到路由页面
2. 选择“对等体”选项卡

3. 点击“添加新对等体”
4. 选择“M2PA (RFC 4165)”从协议下拉菜单
5. 填写对等体配置：
    - 对等体名称（描述性标识符）
    - 协议：M2PA
    - 角色：客户端或服务器
    - 点码（您的 PC）
    - 本地/远程 IP 地址
    - 本地/远程端口（通常为 3565 的 M2PA）
    - 网络指示符（国际或国家）
6. 点击“保存对等体”

对等体表显示协议类型并进行颜色编码：

- 蓝色 - M3UA 对等体
- 绿色 - M2PA 对等体

M2PA 路由行为

M2PA 对等体与 OmniSS7 的路由系统无缝集成：

- 点码路由：对于 M2PA 和 M3UA 工作方式相同
- 全局标题路由：在 M2PA 链接上完全支持
- 路由优先级：M2PA 和 M3UA 对等体可以混合在同一路由表中
- 消息中继：消息可以在 M2PA 上到达并路由到 M3UA，反之亦然

M2PA □标

M2PA 提供全面的 Prometheus 指标，用于监控链路健康和流量：

流量□标：

- `m2pa_messages_sent_total` - 每个链路发送的总 MTP3 消息
- `m2pa_messages_received_total` - 每个链路接收的总 MTP3 消息
- `m2pa_bytes_sent_total` - 通过 M2PA 发送的总字节
- `m2pa_bytes_received_total` - 通过 M2PA 接收的总字节

所有流量指标按：`link_name`、`point_code`、`adjacent_pc` 标记

链路状态指标：

- `m2pa_link_state_changes_total` - 链路状态转换（DOWN → ALIGNMENT → PROVING → READY）
  - 标签：`link_name`、`from_state`、`to_state`

错误指标：

- `m2pa_errors_total` - 按类型统计的总错误
  - `decode_error` - M2PA 消息解码失败
  - `encode_error` - M2PA 消息编码失败
  - `sctp_send_error` - SCTP 传输失败
  - 标签：`link_name`、`error_type`

访问指标：

- Prometheus 端点：`http://your-server:8080/metrics`
- 指标在应用程序启动时自动注册

M2PA 最佳实践

1. 端口选择：使用端口 3565 作为 M2PA（行业标准）
2. 链路监控：通过指标监控链路状态变化
3. 防火墙规则：确保 SCTP（IP 协议 132）被允许
4. 点码：确保相邻点码在两侧正确配置
5. 网络指示符：必须在对等体之间匹配（国际或国家）
6. 测试：使用路由测试页面在配置后验证连接

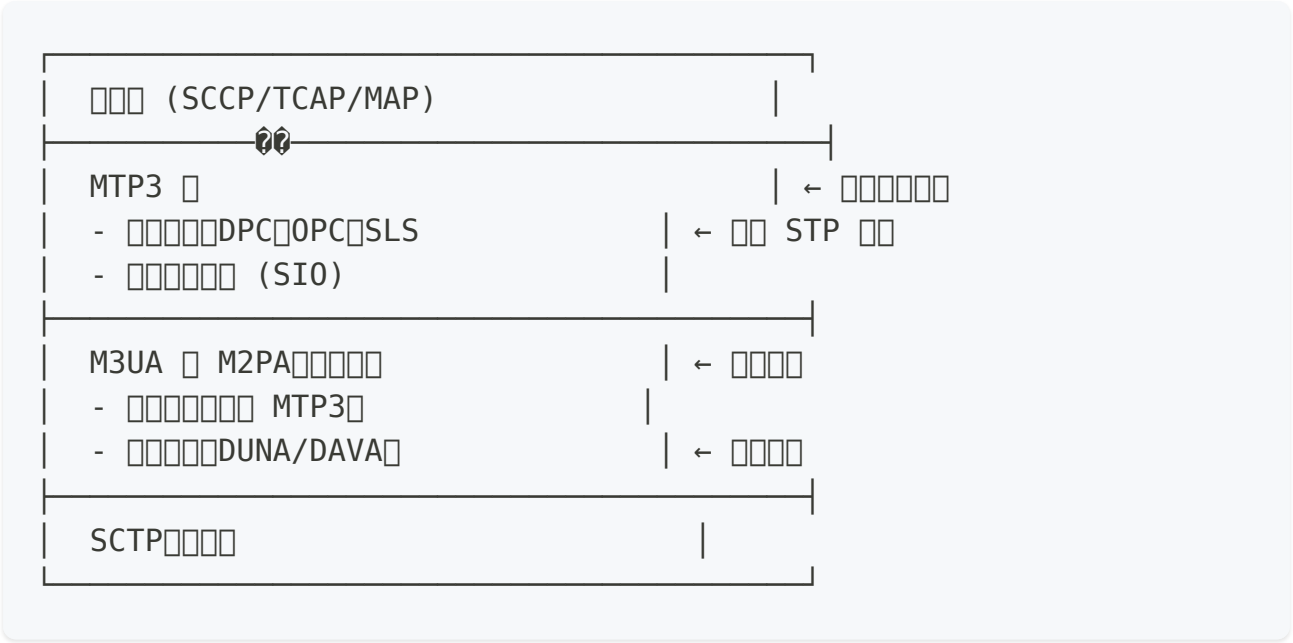
配置点码路由

点码路由根据 MTP3 头中的 目标点码 **(DPC)** 指导消息。

□□ SS7 □□□□□□□□

□□□□□□ SS7 □□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□



□□□□□□□□□□

1. **MTP3** □□□□□□□□□□□□□□
   - □□ MTP3 □□□□□□□□DPC□OPC□
   - □□□□ M3UA □□□□□□□□□□□□ 528□□
   - □□□□ M2PA □□□□□□□□□
   - **STP** □□□□□ **DPC** □□□□□□□□□
   - □□□□□□□□□□□□□□□□□□□□
2. **M3UA** □□□□□□□□□□□□□□□□
   - □□□□ M3UA □□□□□□□□DUNA□DAVA□SCON□DUPU□
   - □□□□□□□□□□□□□□□□□□□□
   - □□□□□□□□□□□□□□□□/□□□□
   - □□□□□□□□□□□□

STP □□□□□□□□□

- 对于 **M3UA** 数据消息：STP 从协议数据参数（标签 528）中提取 MTP3 消息，该消息包含 MTP3 路由标签（DPC、OPC、SLS）。MTP3 层的 DPC 用于查找路由。
- 对于 **M2PA** 用户数据消息：STP 从 M2PA 用户数据字段中提取 MTP3 消息，然后读取 MTP3 路由标签中的 DPC。
- **M3UA** 管理消息：网络管理消息（DUNA、DAVA、SCON）在 M3UA 层中包含受影响的点码，用于在对等体之间信令网络状态。

基本点码路由

将路由添加到 `config/runtime.exs`：

```
config :omniss7,
  m3ua_routes: [
    # 将所有流量路由到 PC 100 的对等体 1（合作伙伴 STP）
    %{
      dest_pc: 100,                    # 目标点码
      peer_id: 1,                      # 通过的对等体
      priority: 1,                     # 优先级（数字越小优先级越高）
      network_indicator: :international
      # mask: 14                       # 可选：默认为 14（精确匹配）
    },

    # 将所有流量路由到 PC 200 的对等体 2（本地 HLR）
    %{
      dest_pc: 200,
      peer_id: 2,
      priority: 1,
      network_indicator: :international
    },

    # 负载均衡示例：PC 300 的主路由和备份路由
    %{
      dest_pc: 300,
      peer_id: 3,                      # 主路由
      priority: 1,
      network_indicator: :international
    },
    %{
      dest_pc: 300,
      peer_id: 4,                      # 备份路由（优先级数字更高）
      priority: 2,
      network_indicator: :international
    }
  ]
```

注意：`mask` 字段是可选的，默认为 `14`（精确点码匹配）。仅在需要基于范围的路由时指定 `mask`（请参见点码掩码部分）。

路由逻辑

1. STP 接收 M3UA 数据或 M2PA 用户数据消息
2. STP 从协议数据（M3UA）或用户数据（M2PA）字段提取 **MTP3** 消息
3. STP 从 MTP3 路由标签中读取目标点码 **(DPC)**

4. 查找路由表以匹配 DPC（考虑掩码）
5. 如果存在多个路由，选择具有 最具体掩码（最高掩码值）的路由，然后 最低优先级数字
6. 将 MTP3 消息封装在 M3UA 数据或 M2PA 用户数据中以发送到目标对等体
7. 将消息路由到相应的对等体
8. 如果所选对等体处于关闭状态，则尝试下一个优先级最高的路由

点码掩码

点码是 14 位值（范围 0-16383）。默认情况下，路由���确匹配单个点码（掩码 `/14`）。但是，您可以使用 点码掩码 创建匹配点码范围 的路由。

理解掩码

掩码指定路由的目标 PC 和传入消息的 DPC 之间必须匹配的 最重要的位数。其余位可以是任何值，从而创建匹配点码的范围。

掩码参考表：

掩码	匹配的点码	用例
`/14`	1 PC（精确匹配）	单一目标（默认）
`/13`	2 PCs	小范围
`/12`	4 PCs	小范围
`/11`	8 PCs	小范围
`/10`	16 PCs	中等范围
`/9`	32 PCs	中等范围
`/8`	64 PCs	中等范围
`/7`	128 PCs	中大范围
`/6`	256 PCs	大范围
`/5`	512 PCs	大范围
`/4`	1,024 PCs	非常大范围
`/3`	2,048 PCs	非常大范围
`/2`	4,096 PCs	极大范围
`/1`	8,192 PCs	一半的所有 PCs
`/0`	16,384 PCs	所有 PCs（默认/回退路由）

点码掩码示例

注意：`mask` 字段是 可选的，在所有示例中。如果省略，默认为 `14`（精确匹配）。

示例 **1**：单个点码（默认行为）

```
# □□□□□□□□□□□□□□□□ PC□
%{
  dest_pc: 1000,
  peer_id: 1,
  priority: 1,
  network_indicator: :international
}
# □□□□□ 14 - □□□□□ PC 1000

# □□□□□□□□□□□□
%{
  dest_pc: 1000,
  peer_id: 1,
  priority: 1,
  mask: 14,                                # □□□□□□□
  network_indicator: :international
}
# □□□□□ PC 1000
```

□□ 2□□□□

```
%{
  dest_pc: 1000,
  peer_id: 2,
  priority: 1,
  mask: 12,                                # □□ 4 PCs
  network_indicator: :international
}
# □□□PC 1000□1001□1002□1003
```

□□ 3□□□□□

```
%{
  dest_pc: 1000,
  peer_id: 3,
  priority: 1,
  mask: 8,                              # 匹配 64 PCs
  network_indicator: :international
}
# 匹配: PC 1000-1063（64 个连续点码）
```

示例 **4**：默认/回退路由

```
%{
  dest_pc: 0,
  peer_id: 4,
  priority: 10,                         # 低优先级（高数字）
  mask: 0,                              # 匹配所有 PCs
  network_indicator: :international
}
# 匹配: 所有点码（0-16383）
# 用作低优先级的捕获所有/默认路由
```

结合特定和掩码路由

您可以结合特定路由和掩码路由以实现灵活路由：

```
config :omniss7,
  m3ua_routes: [
    # 针对 PC 1000 的特定路由（优先级最高）
    %{
      dest_pc: 1000,
      peer_id: 1,
      priority: 1,
      network_indicator: :international
      # 掩码默认为 14（精确匹配）
    },

    # 针对 PCs 1000-1063 的范围路由
    %{
      dest_pc: 1000,
      peer_id: 2,
      priority: 1,
      mask: 8,                           # 匹配 64 PCs
      network_indicator: :international
    },

    # 针对所有其他 PCs 的默认/回退路由
    %{
      dest_pc: 0,
      peer_id: 3,
      priority: 10,                      # 低优先级
      mask: 0,                           # 匹配所有 PCs
      network_indicator: :international
    }
  ]
```

DPC 1000 的路由决策：

1. 匹配掩码 `/14` 路由（PC 1000 精确） - 选择（最具体）
2. 也匹配掩码 `/8` 路由（PC 1000-1063 范围） - 被忽略（不够具体）
3. 也匹配掩码 `/0` 路由（所有 PCs） - 被忽略（最不具体）

DPC 1015 的路由决策：

1. 不匹配掩码 `/14` 路由（仅 PC 1000）
2. 匹配掩码 `/8` 路由（PC 1000-1063 范围） - 选择（最具体匹配）
3. 也匹配掩码 `/0` 路由（所有 PCs） - 被忽略（最不具体）

DPC 5000 的路由决策：

1. 不匹配掩码 `/14` 路由
2. 不匹配掩码 `/8` 路由
3. 匹配掩码 `/0` 路由（所有 PCs） - 选择（唯一匹配，回退路由）

最佳实践

1. 省略 **`mask`** 用于单个目标：对于精确点码匹配，完全省略 `mask` 字段（默认为 `/14`）
2. 仅在需要时显式使用 **`/14`**：仅在需要在文档中明确说明或与范围路由混合时指定 `mask: 14`
3. 使用范围掩码用于网络块：将整个网络段路由到特定对等体，使用掩码 `/0` 到 `/13`
4. 使用 **`/0`** 作为回退：创建一个低优先级的默认路由以捕获未匹配的流量
5. 最具体的优先：路由引擎始终首先选择最具体（最高掩码值）匹配的路由
6. 优先级作为平局打破者：如果多个路由具有相同掩码，则最低优先级数字优先

配置全局标题 **(GT)** 路由

全局标题路由启用基于内容的路由，使用电话号码或 IMSI 值而不是点码。有关基于呼叫方/被叫方的高级全局标题地址转换，请参见 全局标题 NAT 指南。

前提条件

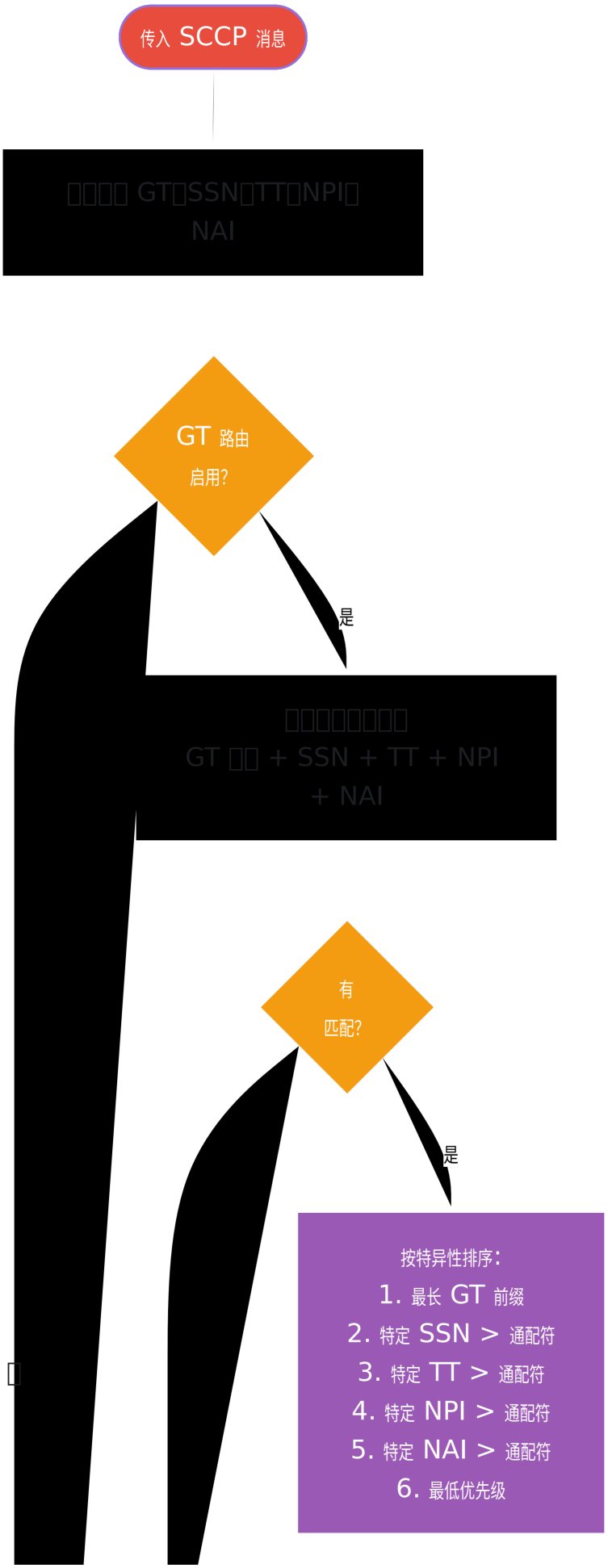
- 启用 GT 路由：在 `config/runtime.exs` 中设置 `enable_gt_routing: true`

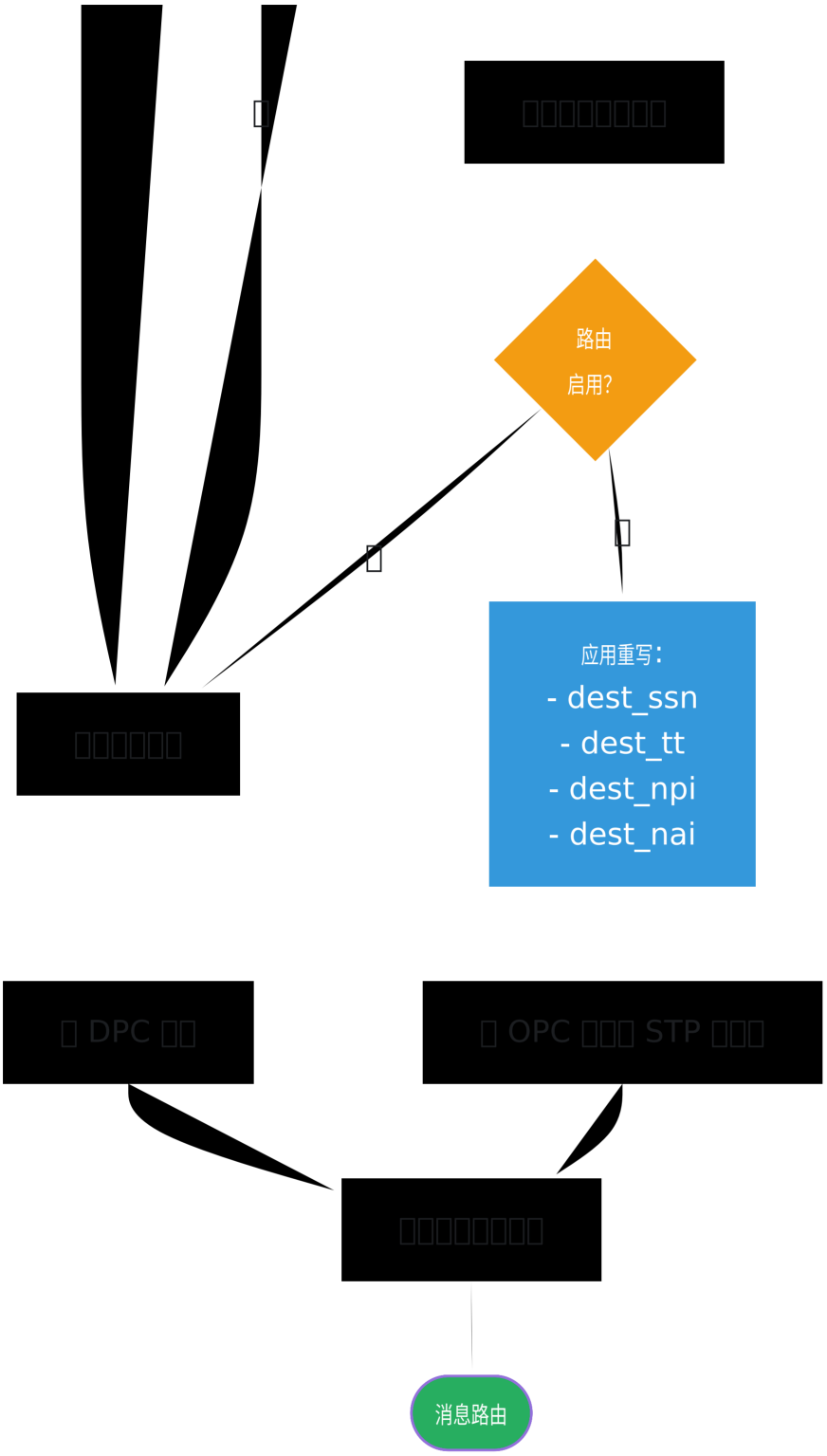
GT 路由配置

```
config :omniss7,
  # 启用 GT 路由
  enable_gt_routing: true,

  m3ua_gt_routes: [
    # 将所有英国号码（前缀 44）路由到对等体 1
    %{
      gt_prefix: "44",                 # 匹配的全局标题前缀
      peer_id: 1,                      # 目标对等体
      priority: 1,                     # 优先级（数字越小优先级越高）
      description: "英国号码"          # 日志描述
    },

    # 将美国号码（前缀 1）路由到对等体 2
    %{
      gt_prefix: "1",
      peer_id: 2,
      priority: 1,
      description: "美国号码"
    },

    # 更具体的路由：以 447 开头的英国手机号码
    %{
      gt_prefix: "447",                # 最长前缀匹配胜出
      peer_id: 3,
      priority: 1,
      description: "英国手机号码"
    },

    # SSN 特定路由（可选）
    %{
      gt_prefix: "555",
      source_ssn: 8,                   # 仅在源 SSN = 8（SMSC）时匹配
      peer_id: 4,
      dest_ssn: 6,                     # 将目标 SSN 重写为 6（HLR）
      priority: 1,
      description: "以 61 前缀的 SMS 流量"
    }
  ]
```

GT 路由逻辑

GT 路由算法遵循以下决策过程：


传入 SCCP 消息
GT 路由
启用?
是
有
匹配?
是
按特异性排序:
1. 最长 GT 前缀
2. 特定 SSN > 通配符
3. 特定 TT > 通配符
4. 特定 NPI > 通配符
5. 特定 NAI > 通配符
6. 最低优先级



路由步骤:

1. 最长前缀匹配: STP 查找所有 GT 路由，其中前缀与全局标题的开头匹配
    - 示例: GT "447712345678" 匹配 "44" 和 "447"，但 "447" 胜出（最长匹配）
2. **SSN** 匹配（可选）:

- 如果指定了 `source_ssn`，则仅在 SCCP 被叫方 SSN 等于该值时路由匹配
- 如果 `source_ssn` 为 `nil`，则路由匹配任何 SSN（通配符）

3. **TT/NPI/NAI** 匹配（可选）：

- 如果指定了 `source_tt`、`source_npi` 或 `source_nai`，则路由必须匹配这些指示符
- `nil` 值作为通配符（匹配任何值）

4. 基于特异性的选择：

- 具有更具体匹配条件的路由胜出通配符
- 优先级顺序：GT 前缀长度 → SSN → TT → NPI → NAI → 优先级数字

5. □示符重写（可选）：

- 如果指定了 `dest_ssn`、`dest_tt`、`dest_npi` 或 `dest_nai`，STP 将重写这些指示符

- 对于协议规范化和网络互连非常有用

6. 回退到点码：
    - 如果没有 GT 路由匹配，STP 将回退到使用 DPC 的点码路由

高级 **GT** 路由：转换类型、**NPI** 和 **NAI**

除了 GT 前缀和 SSN 匹配外，STP 还支持基于 SCCP 全局标题指示符的路由和转换：

- 转换类型 **(TT)**：标识编号计划和地址类型
- 编号计划□示符 **(NPI)**：定义编号计划（例如，ISDN、数据、传真）
- 地址类型□示符 **(NAI)**：指定地址格式（例如，国际、国家、用户）

匹配（源□示符）

路由可以根据传入消息指示符进行匹配：

- `source_tt`：匹配具有特定转换类型的消息
- `source_npi`：匹配具有特定编号计划指示符的消息
- `source_nai`：匹配具有特定地址类型指示符的消息
- `nil` 值 = 通配符（匹配任何值）

转换（目标□示符）

路由在转发时可以重写指示符：

- `dest_tt`：将转换类型转换为新值
- `dest_npi`：将编号计划指示符转换为新值
- `dest_nai`：将地址类型指示符转换为新值
- `nil` 值 = 保留原始值（无转换）

基于特异性的选择

当多个路由匹配时，选择最具体的路由，优先顺序如下：

1. 最长 GT 前缀匹配
2. 特定源 SSN 胜出通配符 SSN
3. 特定源 TT 胜出通配符 TT
4. 特定源 NPI 胜出通配符 NPI
5. 特定源 NAI 胜出通配符 NAI
6. 最低优先级数字

配置示例

```
config :omniss7,
  enable_gt_routing: true,

  m3ua_gt_routes: [
    # 示例 1: 匹配并转换转换类型
    %{
      gt_prefix: "44",
      peer_id: 1,
      source_tt: 0,       # 匹配 TT=0 (未知)
      dest_tt: 3,         # 转换为 TT=3 (国家)
      priority: 1,
      description: "英国号码: TT 0→3 转换"
    },

    # 示例 2: 匹配特定 NPI 并转换 NAI
    %{
      gt_prefix: "1",
      peer_id: 2,
      source_npi: 1,      # 匹配 NPI=1 (ISDN/电信)
      source_nai: 4,      # 匹配 NAI=4 (国际)
      dest_nai: 3,        # 转换为 NAI=3 (国家)
      priority: 1,
      description: "美国号码: 国际→国家 NAI"
    },

    # 示例 3: 结合 SSN 和指示符路由
    %{
      gt_prefix: "33",
      source_ssn: 8,      # 匹配 SMSC 流量
      source_tt: 0,       # 匹配 TT=0
      dest_ssn: 6,        # 将 SSN 重写为 HLR
      dest_tt: 2,         # 转换为 TT=2
      dest_npi: 1,        # 设置 NPI=1 (ISDN)
      dest_nai: 4,        # 设置 NAI=4 (国际)
      peer_id: 3,
      priority: 1,
      description: "法国 SMS: 完全规范化"
    },

    # 示例 4: 通配符 TT, 特定 NPI
    %{
      gt_prefix: "49",
      source_tt: nil,     # 匹配任何 TT (通配符)
```

```
        source_npi: 6,      # 匹配 NPI=6（数据）
        dest_npi: 1,        # 转换为 NPI=1（ISDN）
        peer_id: 4,
        priority: 1,
        description: "德国数据网络规范化"
      }
    ]
```

常见 **TT/NPI/NAI** 值

转换类型 **(TT)**:

- `0` = 未知
- `1` = 国际
- `2` = 国家
- `3` = 网络特定

编号计划□示符 **(NPI)**:

- `0` = 未知
- `1` = ISDN/电信（E.164）
- `3` = 数据（X.121）
- `4` = 传真（F.69）
- `6` = 陆地移动（E.212）

地址类型□示符 **(NAI)**:

- `0` = 未知
- `1` = 用户号码
- `2` = 保留用于国家使用
- `3` = 国家显著号码
- `4` = 国际号码

路由决策示例

对于传入消息:

- GT: "447712345678"

- SSN: 8
- TT: 0
- NPI: 1
- NAI: 4

配置了这些路由：

```
# 路由 A: 通配符 TT
%{gt_prefix: "447", peer_id: 1, priority: 1}

# 路由 B: 特定 TT
%{gt_prefix: "447", source_tt: 0, peer_id: 2, priority: 1}

# 路由 C: 特定 TT + NPI
%{gt_prefix: "447", source_tt: 0, source_npi: 1, peer_id: 3,
priority: 1}
```

结果：路由 C 被选择（最具体：匹配 GT + TT + NPI）

消息根据路由 C 的 `dest_tt`、`dest_npi`、`dest_nai` 值进行转发。

GT 路由示例

被叫 GT	源 SSN	TT	NPI	NAI	匹配的路由	原因
447712345678	6	-	-	-	"447" → 对等体 3	最长前缀匹配
441234567890	6	-	-	-	"44" → 对等体 1	前缀匹配，没有更具体的路由
12125551234	6	-	-	-	"1" → 对等体 2	美国号码的前缀匹配
555881234567	8	-	-	-	"555" (SSN 8) → 对等体 4	GT + SSN 匹配，重写 SSN 为 6
555881234567	6	-	-	-	"555" (SSN 通配符) → 对等体 X	GT 匹配，没有 SSN 重写
441234567890	6	0	1	4	"44" (TT=0) → 对等体 1	GT + TT 匹配，转换 TT 为 3
12125551234	8	0	1	4	"1" (TT=0, NPI=1, NAI=4)	最具体: GT+TT+NPI+NAI 匹配

TT/NPI/NAI 路由的实际用例

1. 网络互连规范化
    - 不同网络可能使用不同的指示符约定
    - 在互连点转换指示符以确保兼容性
    - 示例：合作伙伴网络使用 TT=0 表示国际，您的网络使用 TT=1
2. 协议转换
    - 在不同网络类型之间路由时转换编号计划

- □□□□□□□□□□NPI=6□□□□ PSTN□NPI=1□

3. □□□□□□□

- □□□□□□□□□□□□□□□□ NAI □
- □□□□□□□□□□□□NAI=4□□□□□□□□□□NAI=3□□□□□□□□

4. □□□□□□□

- □□□□□□□□□□□□□□□□□□□□□□□□
- □□□TT=0 □□□□□□ A□TT=2 □□□□□□ B

5. □□□□□□

- □□□□□□□□□□□□□□□□□□□□□□□□
- □ STP □□□□□□□□□□□

□□□□□

□□□

□□□□□□□□□□□□□□□□□□□

```
config :omniss7,
  m3ua_routes: [
    # 活动路由
    %{
      dest_pc: 100,
      peer_id: 1,
      priority: 1,
      network_indicator: :international,
      enabled: true  # 路由处于活动状态（省略时默认为启用）
    },

    # 禁用路由（在路由期间不进行评估）
    %{
      dest_pc: 200,
      peer_id: 2,
      priority: 1,
      network_indicator: :international,
      enabled: false  # 路由已禁用
    }
  ],

  m3ua_gt_routes: [
    # 禁用 GT 路由
    %{
      gt_prefix: "44",
      peer_id: 1,
      priority: 1,
      description: "英国号码 - 暂时禁用",
      enabled: false
    }
  ]
```

默认行为：

- 如果未指定 `enabled`，路由默认为 `enabled: true`
- 禁用的路由在路由查找期间完全跳过
- 使用 Web UI 切换路由的启用/禁用状态，而无需编辑配置

用例：

- 测试流量流向变化

- 临时维护窗口
- A/B 测试不同的路由路径
- 新路由的逐步推出

DROP 路由 - 防止路由循环

DROP 路由（`peer_id: 0`）会静默丢弃流量，而不是转发它。这可以防止路由循环并启用高级流量过滤。

配置 DROP 路由

```
config :omniss7,
  m3ua_routes: [
    # 针对特定点码的 DROP 路由
    %{
      dest_pc: 999,
      peer_id: 0,         # peer_id=0 表示 DROP
      priority: 1,
      network_indicator: :international
    }
  ],

  m3ua_gt_routes: [
    # 针对 GT 前缀的 DROP 路由
    %{
      gt_prefix: "999",
      peer_id: 0,         # peer_id=0 表示 DROP
      priority: 99,
      description: "阻止测试范围"
    }
  ]
```

DROP 路由的工作原理

当消息匹配 DROP 路由时:

1. 路由引擎识别 `peer_id: 0`
2. 消息 被静默丢弃（不转发）

3. 生成信息日志：`"DROP route matched for DPC 999"` 或 `"DROP route matched for GT 999"`
4. 路由查找回 `{:error, :dropped}`

重要：丢弃的流量以信息级别记录，以便于监控和故障排除。

常见用例：前缀白名单

DROP 路由的一个强大用途是 前缀白名单 - 仅允许特定号码在大范围内，同时阻止所有其他号码。

模式：

1. 创建一个针对整个前缀的 DROP 路由，优先级 高（例如，99）
2. 为单个号码创建特定允许路由，优先级 低（例如，1）
3. 由于较低的优先级数字首先被评估，允许的路由在 DROP 路由之前匹配
4. 任何未明确允许的号码都将被 DROP 路由捕获

示例场景：

您有一个 GT 前缀 `1234`，表示 10,000 个号码（1234000000 - 1234999999），但您只想路由 3 个特定号码：`1234567890`、`1234555000` 和 `1234111222`。

```
config :omniss7,
  m3ua_gt_routes: [
    # 使用高优先级数字的 DROP 路由（最后评估）
    %{
      gt_prefix: "1234",
      peer_id: 0,                 # DROP
      priority: 99,               # 高数字 = 低优先级 = 最后评估
      description: "阻止所有 1234* 除了白名单号码"
    },

    # 具有低优先级数字的特定允许路由（优先评估）
    %{
      gt_prefix: "1234567890",
      peer_id: 1,                 # 路由到对等体 1
      priority: 1,                # 低数字 = 高优先级 = 首先评估
      description: "允许的号码 1"
    },
    %{
      gt_prefix: "1234555000",
      peer_id: 1,
      priority: 1,
      description: "允许的号码 2"
    },
    %{
      gt_prefix: "1234111222",
      peer_id: 1,
      priority: 1,
      description: "允许的号码 3"
    }
  ]
```

路由行为：

传入 **GT**	匹配的路由	选择的路由	动作
1234567890	□ "1234567890" (优先级 1) □ "1234" DROP (优先级 99)	"1234567890" (最具体��优先级最高)	路由到对等体 **1**
1234555000	□ "1234555000" (优先级 1) □ "1234" DROP (优先级 99)	"1234555000" (最具体，优先级最高)	路由到对等体 **1**
1234111222	□ "1234111222" (优先级 1) □ "1234" DROP (优先级 99)	"1234111222" (最具体，优先级最高)	路由到对等体 **1**
1234999999	□ "1234" DROP (优先级 99)	"1234" DROP (唯一匹配)	丢弃 + 记录
1234000000	□ "1234" DROP (优先级 99)	"1234" DROP (唯一匹配)	丢弃 + 记录

结果：

- □ □ 3 个特定号码被路由到对等体 1
- □ □□□□ `1234*` 号码被静默丢弃
- □ 所有丢弃的流量被记录以便监控

生成的日志：

```
[INFO] DROP route matched for GT 1234999999
[INFO] DROP route matched for GT 1234000000
```

针对点码的 **DROP** 路由

相同的白名单模式适用于点码路由：

```
config :omniss7,
  m3ua_routes: [
    # DROP 整个范围 /8 (64 个点码: 1000-1063)
    %{
      dest_pc: 1000,
      peer_id: 0,
      priority: 99,
      mask: 8,
      network_indicator: :international
    },

    # 允许特定 PCs
    %{dest_pc: 1010, peer_id: 1, priority: 1, network_indicator:
:international},
    %{dest_pc: 1020, peer_id: 1, priority: 1, network_indicator:
:international},
    %{dest_pc: 1030, peer_id: 1, priority: 1, network_indicator:
:international}
  ]
```

结果: 仅 PCs 1010、1020 和 1030 被路由。所有其他 1000-1063 范围内的 PCs 被丢弃。

监控 **DROP** 路由

检查日志:

```
# 监控丢弃的流量
tail -f logs/app.log | grep "DROP route matched"

# 预期输出:
[INFO] DROP route matched for GT 1234999999
[INFO] DROP route matched for DPC 1050
```

通过 **Web UI**:

- 导航到 系统日志 选项卡
- 按 `INFO` 级别过滤
- 搜索 "DROP route matched"

最佳实践:

1. ⚠ 定期监控日志以确保 DROP 路由不会阻止合法流量
2. □ 使用描述性 `description` 字段记录为何丢弃路由
3. □ 使用高优先级数字（90-99）为 DROP 路由确保它们是捕获所有路由
4. □ □□□□□□□□□□□□ DROP 路由行为
5. □ 设置警报以监控丢弃流量的意外增加

高级路由：基于 **SSN** 的路由和重写

子系统编号 **(SSN)**

子系统编号标识应用层：

- **SSN 6**：HLR（归属位置寄存器）
- **SSN 7**：VLR（访客位置寄存器）
- **SSN 8**：MSC（移动交换中心）/ SMSC（短信中心）
- **SSN 9**：GMLC（网关移动位置中心）

基于 **SSN** ���路由示例

根据号码前缀将 SMS 流量路由到不同的 HLR：

```
m3ua_gt_routes: [
  # 将英国号码的 SMS 路由到英国 HLR，将 SSN 从 8（SMSC）重写为 6（HLR）
  %{
    gt_prefix: "44",
    source_ssn: 8,                    # 匹配传入的 SSN 8（SMSC）
    peer_id: 1,
    dest_ssn: 6,                      # 重写为 SSN 6（HLR）
    priority: 1,
    description: "英国 SMS 到 HLR"
  },

  # 将英国号码的语音流量（SSN 6）路由到同一对等体而不重写
  %{
    gt_prefix: "44",
    source_ssn: 6,                    # 匹配传入的 SSN 6（HLR）
    peer_id: 1,
    dest_ssn: nil,                    # 无 SSN 重写
    priority: 1,
    description: "英国语音流量"
  }
]
```

测试 STP 路由配置

在配置对等体和路由后，验证您的配置：

1. 检查对等体状态

通过 **Web UI**：

- 导航到 http://localhost
- 检查 M3UA 状态页面
- 验证对等体显示 状态：活动

通过 **IEx** 控制台：

```
# 获取所有对等体状态
M3UA.STP.get_peers_status()

# 预期输出:
# [
#   %{peer_id: 1, name: "Partner_STP_West", status: :active,
point_code: 100, ...},
#   %{peer_id: 2, name: "Local_HLR", status: :active, point_code:
200, ...}
# ]
```

2. 测试点码路由

```
# 发送测试 M3UA 消息到 DPC 100
test_payload = <<1, 2, 3, 4>>  # 虚拟负载
M3UA.STP.route_by_pc(100, test_payload, 0)

# 检查日志以获取路由决策
# 预期日志: "Routing message: OPC=... -> DPC=100 via peer 1"
```

3. 测试全局标题路由

```
# 手动查找 GT 路由
M3UARouting.lookup_peer_by_gt("447712345678")

# 预期输出:
# {:ok, {:m3ua_peer, 3, "UK_Mobile_Peer", ...}, nil}

# 带 SSN 查找 GT 路由
M3UARouting.lookup_peer_by_gt("555881234567", 8)

# 预期输出带 SSN 重写:
# {:ok, {:m3ua_peer, 4, "SMS_HLR_Peer", ...}, 6}
```

4. 监控路由□标

访问 Prometheus 指标在 `/metrics`

关键指标:

```
# 每个对等体接收的消息
m3ua_stp_messages_received_total{peer_name="Partner_STP_West",point_c
1523

# 每个对等体发送的消息
m3ua_stp_messages_sent_total{peer_name="Local_HLR",point_code="200"}

# 路由失败
m3ua_stp_routing_failures_total{reason="no_route"} 5
m3ua_stp_routing_failures_total{reason="no_gt_route"} 2
```

STP □标和监控

可用□标

每个对等体的流量□标:

- `m3ua_stp_messages_received_total` - 从每个对等体接收的总消息
  - 标签：`peer_name`，`point_code`
- `m3ua_stp_messages_sent_total` - 转发到每个对等体的总消息
  - 标签：`peer_name`，`point_code`

路由失败□标：

- `m3ua_stp_routing_failures_total` - 按原因统计的路由失败次数
  - 标签：`reason`（值：`no_route`，`no_gt_route`）

□标解释

- 高消息计数：表示活跃的流量流动
- 路由失败：表示缺少路由或配置错误
  - `no_route`：未找到目标点码路由
  - `no_gt_route`：未找到全局标题路由，且 PC 路由也失败

使用□标进行故障排除

场景：没有流量到达目标

1. 检查消息是否被接收：

```
m3ua_stp_messages_received_total{peer_name="Source_Peer"} > 0
```

2. 检查消息是否被发送：

```
m3ua_stp_messages_sent_total{peer_name="Dest_Peer"} > 0
```

3. 检查路由失败：

```
m3ua_stp_routing_failures_total{reason="no_route"} > 0
```

解决方案：如果路由失败高，添加缺失的路由到配置中。

M3UA 对等体状态监控

理解 M3UA

M3UA（MTP3 用户适配层）是一种协议，允许 SS7 信令通过 IP 网络使用 SCTP 进行传输。

M3UA 连接状态

M3UA 连接经历几个状态：

Web UI 指南

← 返回主文档

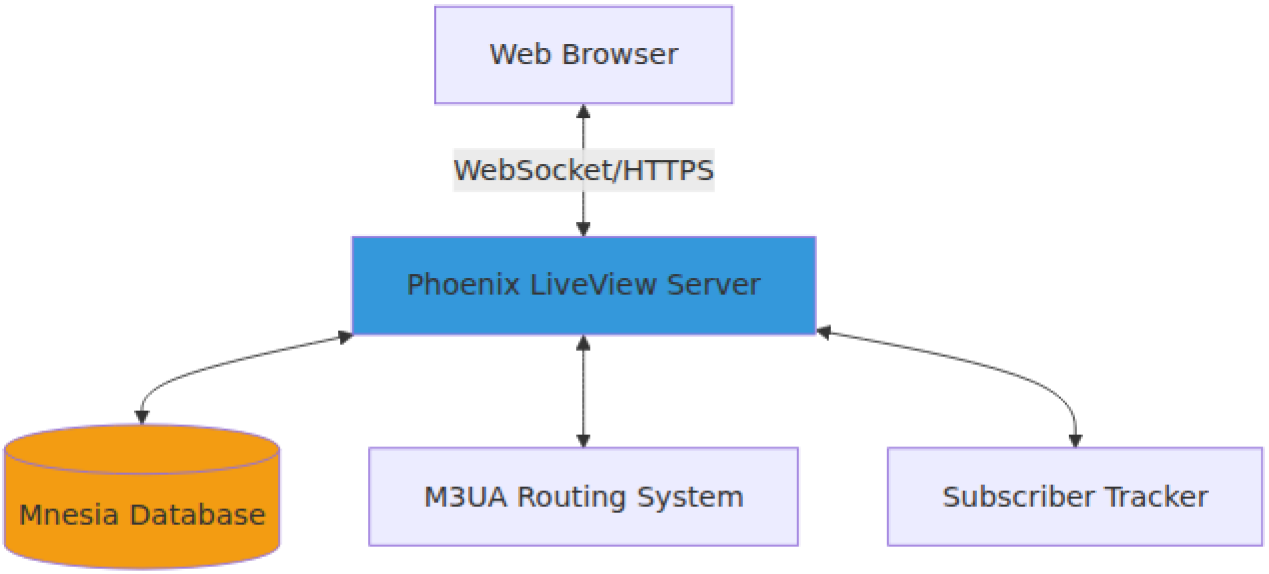
本指南提供了使用 OmniSS7 **Web UI**（Phoenix LiveView 接口）的全面文档。

目录

概述

OmniSS7 Web UI 是一个 **Phoenix LiveView** 应用程序，提供实时监控和管理功能。可用页面取决于当前激活的操作模式（STP、HLR 或 SMSc）。

Web UI 架构



服务器配置

- 协议: HTTPS
- 端口: 443 (在 `config/runtime.exs` 中配置)
- 默认 **IP**: 0.0.0.0 (监听所有接口)
- 证书: 位于 `priv/cert/`

访问 **URL**: `https://[server-ip]:443`

访问 Web UI

先决条件

1. **SSL** 证书: 确保 `priv/cert/` 中存在有效的 SSL 证书:
   - `omnitouch.crt` - 证书文件
   - `omnitouch.pem` - 私钥文件
2. 应用程序运行: 使用 `iex -S mix` 启动应用程序
3. 防火墙: 确保端口 443 对 HTTPS 流量开放

按模式可用页面

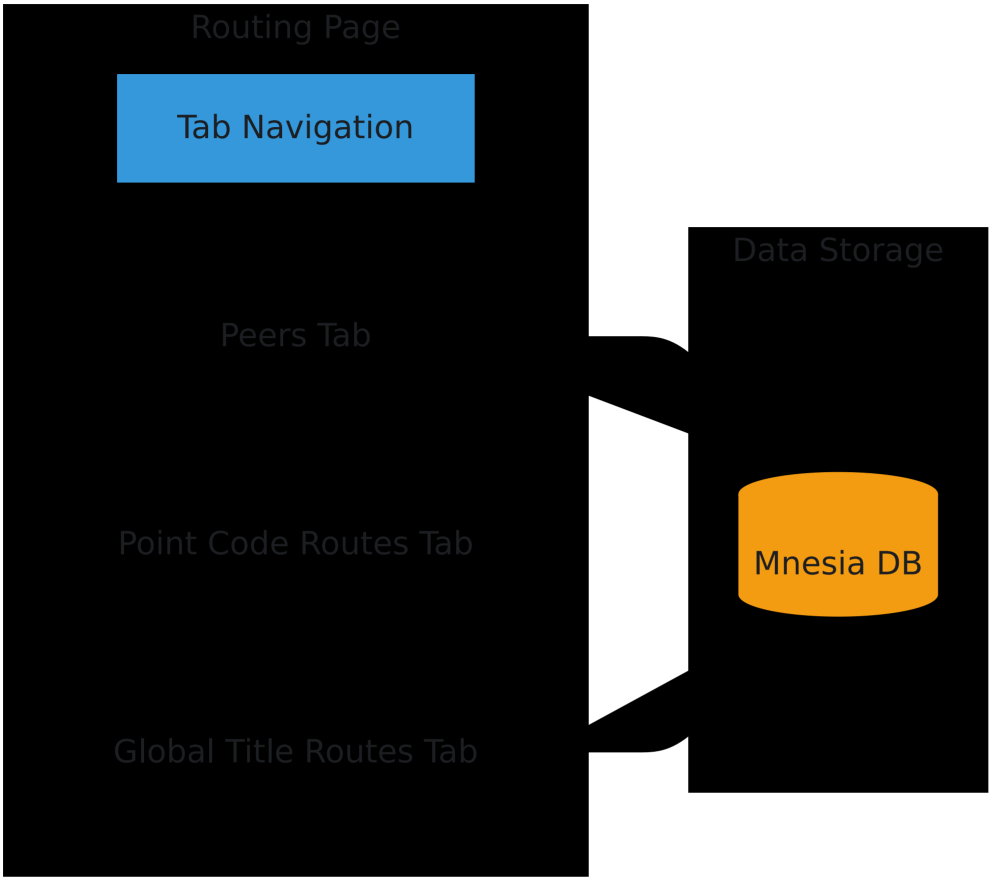
页面	**STP** 模式	**HLR** 模式	**SMSc** 模式	描述
SS7 事件	▯	▯	▯	事件日志和 SCCP 消息捕获
SS7 客户端	▯	▯	▯	手动 MAP 操作测试
M3UA	▯	▯	▯	M3UA 连接状态
路由	▯	▯	▯	M3UA 路由表管理
路由测试	▯	▯	▯	路由测试和验证
HLR 链接	▯	▯	▯	HLR API 状态和用户管理
活动用户	▯	▯	▯	实时用户位置跟踪（HLR）
SMSc 链接	▯	▯	▯	SMSc API 状态和队列管理
SMSc 用户	▯	▯	▯	实时用户跟踪（SMSc）
应用程序	▯	▯	▯	系统资源和监控
配置	▯	▯	▯	配置查看器

路由管理

页面：`/routing`
模式：STP, SMSc
自动刷新：每 5 秒

路由管理页面提供了一个选项卡界面，用于管理 M3UA 路由表。

页面布局



Peers 选项卡

管理 M3UA 对等连接（其他 STP、HLR、MSC、SMSC）。

对等表列

列	描述	示例
ID	唯一对等标识符	`1`
名称	可读的对等名称	`"STP_West"`
角色	连接角色	`client`, `server`, `stp`
点代码	对等的 SS7 点代码	`100`
远程	远程 IP:端口	`10.0.0.10:2905`
状态	连接状态	`active`, `aspup`, `down`
操作	编辑/删除按钮	-

添加对等

1. 点击 Peers 选项卡
2. 填写 表单字段:
    - 对等 **ID**: 如果留空则自动生成
    - 对等名称: 描述性名称（必填）
    - 角色: 选择 `client`, `server` 或 `stp`
    - 点代码: SS7 点代码（必填）
    - 本地 **IP**: 您系统的 IP 地址
    - 本地端口: 动态端口分配为 0
    - 远程 **IP**: 对等的 IP 地址
    - 远程端口: 对等的端口（通常为 2905）
    - 路由上下文: M3UA 路由上下文 ID
    - 网络□示符: `international` 或 `national`
3. 点击 "Add Peer"

持久性: 对等信息立即保存到 Mnesia，并在重启后仍然存在。

编辑对等

1. 点击 对等行上的 "Edit" 按钮
2. 根据需要修改 表单字段
3. 点击 "Update Peer"

注意：如果您更改对等 ID，旧的对等将被删除并创建一个新的。

删除对等

1. 点击 对等行上的 "Delete" 按钮
2. 确认 删除（所有使用此对等的路由也将被移除）

对等状态□示器

状态	颜色	描述
`active`	□ 绿色	对等已连接并路由消息
`aspup`	□ 黄色	ASP 已启动但尚未激活
`down`	□ 红色	对等已断开连接

点代码路由选项卡

根据目标点代码配置路由规则。

路由表列

列	描述	示例
目标 **PC**	目标点代码（zone.area.id 格式）	`1.2.3` (100)
掩码	用于 PC 匹配的子网码	`/14` (精确), `/8` (范围)
对等 **ID**	此路由的目标对等	`1`
对等名称	目标对等的名称	`"STP_West"`
优先级	路由优先级（1 = 最高）	`1`
网络	网络指示符	`international`
操作	编辑/删除按钮	-

添加点代码路由

1. 点击 "Point Code Routes" 选项卡
2. 填写 表单字段：
   - 目标点代码：输入为 `zone.area.id`（例如，`1.2.3`）或整数（0-16383）
   - 掩码：选择掩码 `/14` 进行精确匹配，较小值用于范围
   - 对等 **ID**：从下拉菜单中选择目标对等
   - 优先级：输入优先级（1 = 最高，默认）
   - 网络示符：选择 `international` 或 `national`
3. 点击 "Add Route"

点代码格式：您可以以两种格式输入点代码：

- **3-8-3** 格式：`zone.area.id`（例如，`1.2.3`）
- 整数格式：`0-16383`（例如，`1100`）

系统会自动在格式之间转换。

理解掩码

点代码是 14 位值（0-16383）。掩码指定必须匹配的最高有效位数：

掩码	匹配的 **PC** 数	用例
`/14`	1 (精确匹配)	路由到特定目的地
`/13`	2 PCs	小范围
`/8`	64 PCs	中等范围
`/0`	所有 16,384 PCs	默认/后备路由

示例:

- `PC 1000 /14` → 仅匹配 PC 1000
- `PC 1000 /8` → 匹配 PC 1000-1063（64 个连续的 PC）
- `PC 0 /0` → 匹配所有点代码（默认路由）

点代码掩码参考卡

网页包含一个交互式参考，显示所有掩码值及其范围。

全球标题路由选项卡

根据 SCCP 全球标题地址配置路由规则。

要求: 必须在配置中启用全球标题路由:

```
config :omniss7,
  enable_gt_routing: true
```

路由表列

列	描述	示例
GT 前缀	被叫方 GT 前缀（空 = 后备）	`"1234"`, `""`
源 **SSN**	匹配被叫方 SSN（可选）	`6` (HLR), `any`
对等 **ID**	目标对等	`1`
对等	对等名称	`"HLR_West (1)"`
目标 **SSN**	转发时重写 SSN（可选）	`6`, `preserve`
优先级	路由优先级	`1`
描述	路由描述	`"US numbers"`
操作	编辑/删除按钮	-

添加全球标题路由

1. 点击 "Global Title Routes" 选项卡
2. 填写 表单字段：
    - **GT** 前缀：留空以用于后备路由，或输入数字（例如，`"1234"`）
    - 源 **SSN**：可选 - 按被叫方 SSN 过滤
    - 对等 **ID**：选择目标对等
    - 目标 **SSN**：可选 - 转发时重写 SSN
    - 优先级：路由优先级（1 = 最高）
    - 描述：可读描述
3. 点击 "Add Route"

后备路由：如果 GT 前缀为空，则该路由充当不匹配任何其他路由的 GT 的捕获。

常见 **SSN** 值

页面包含一个参考卡，列出常见的 SSN 值：

SSN	网络元素
6	HLR (家庭位置寄存器)
7	VLR (访客位置寄存器)
8	MSC (移动交换中心)
9	EIR (设备身份寄存器)
10	AUC (认证中心)
142	RANAP
145	gsmSCF (服务控制功能)
146	SGSN

SSN 重写

- 源 **SSN**: 匹配传入消息中的被叫方 SSN
- 目标 **SSN**: 如果设置，在转发时重写被叫方 SSN
  - 空 = 保留原始 SSN
  - 值 = 替换为此 SSN

用例：将 SSN=6（HLR）的消息路由到对等，并在出站时重写为 SSN=7（VLR）。

路由表持久性

所有路由都存储在 **Mnesia** 中，并在应用程序重启后仍然存在。

路由如何持久化

1. **Web UI** 更改：所有添加/编辑/删除操作立即保存到 Mnesia
2. 应用程序重启：启动时从 Mnesia 加载路由
3. **Runtime.exs** 合并：从 `config/runtime.exs` 中的静态路由与 Mnesia 路由合并（无重复）

路由优先级

当多个路由匹配一个目标时：

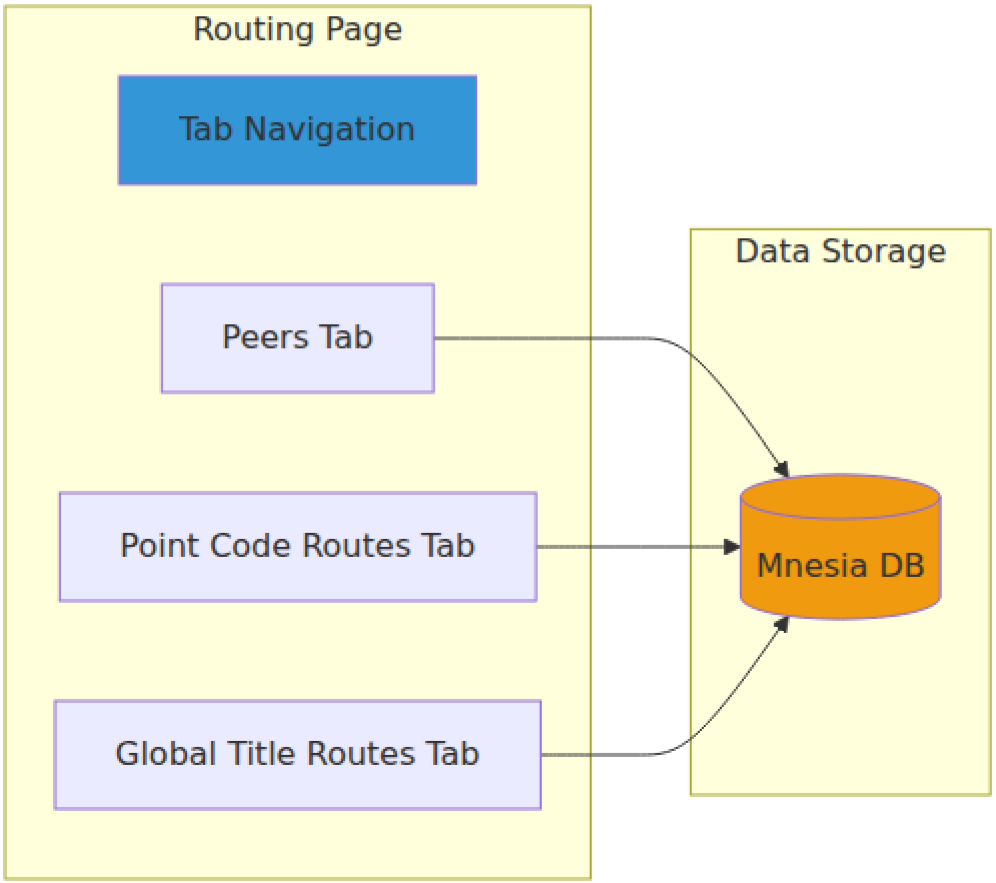
1. 更具体的优先：更高的掩码值（更具体）优先
2. 优先级字段：较低的优先级数字优先路由（1 = 最高优先级）
3. 对等状态：仅使用到 `active` 对等的路由

活动用户

页面：`/subscribers`
模式：仅 HLR
自动刷新：每 2 秒

显示实时跟踪发送 UpdateLocation 请求的用户。

页面功能



□□□□

□	□□	□□
IMSI	□□ IMSI	`"50557123456789"`
VLR □□	□□ VLR GT □□	`"555123155"`
MSC □□	□□ MSC GT □□	`"555123155"`
□□□□	□□ UpdateLocation □□□	`"2025-10-25 14:23:45 UTC"`
□□□□	□□□□□□□□□	`"2h 15m 34s"`

□□□□

□□□□□□□□□□□□□□□□

- **□□□**□□: □□□□□□□□
- □□ **VLRs**: □□ VLR □□□□□
- □□ **MSCs**: □□ MSC □□□□□

□□□□

□□□□□□□: □□□□□□□□□□□□□□□□□

□□: □□□□□□□□□□□□□□□□□

□□: □□□□□□□□□□□□□□□□□□□□□□□□

□□□□

□□□ **2** □ □□□□□□□□□□□□□□□□□

SMSc 用户

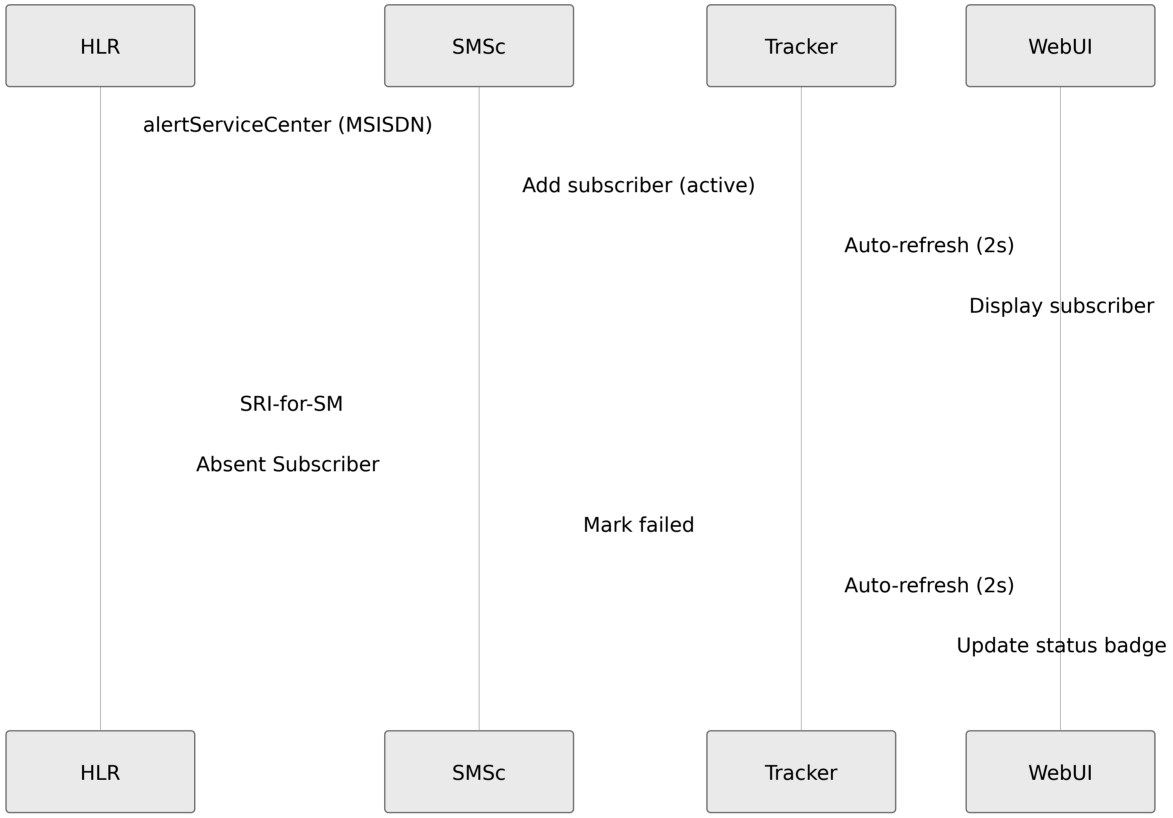
页面: `/smsc_subscribers`

模式: 仅 SMSc

自动刷新: 每 2 秒

显示基于从 HLR 接收到的 alertServiceCenter 消息、消息传递状态和失败跟踪的用户实时跟踪。

页面功能



用户表列

列	描述	示例
MSISDN	用户的电话号码	`"15551234567"`
IMSI	用户 IMSI	`"001010123456789"`
HLR GT	发送 alertServiceCenter 的 HLR GT	`"15551111111"`
发送的消息	发送的 MT-FSM 消息计数	`5`
接收的消息	从用户接收的 MO-FSM 消息计数	`2`
状态	`Active` 或 `Failed`（颜色编码）	`● Active`
最后更新时间	最后更新时间戳	`"2025-10-30 14:23:45 UTC"`
持续时间	自最后更新以来的时间	`"15m 34s"`

状态□示器

- ● **Active** (绿色): 用户可达，最后一次成功接收 alertServiceCenter
- ○ **Failed** (红色): 最后一次投递尝试失败（SRI-for-SM 或缺失用户错误）

统计摘要

当用户存在时，摘要卡显示：

- 总跟踪：跟踪的用户总数
- 活动：活动状态的用户数量
- 失败：失败状态的用户数量
- 唯一 **HLRs**: 发送警报的不同 HLR 数量

管理用户

移除按钮：从跟踪中移除单个用户。

清除所有按钮：移除所有跟踪的用户。

确认：清除所有需要确认（无法撤销）。

用例：

- 在网络问题后移除过期条目
- 在开发后清除测试数据
- 监控哪些 HLR 正在发送警报

消息计数器

跟踪器会自动增加计数器：

- 发送的消息：当 SRI-for-SM 成功并发送 MT-FSM 时增加
- 接收的消息：当从用户接收 MO-FSM 时增加

自动刷新

页面每 **2** 秒自动刷新，以显示实时用户和状态更新。

常见操作

搜索和过滤

目前，Web UI 不包括内置的搜索/过滤功能。要查找特定路由：

1. 使用浏览器的查找功能 (Ctrl+F / Cmd+F)
2. 搜索对等名称、点代码或 GT 前缀

批量操作

要执行批量路由更改：

1. 选项 **1**：使用 REST API 进行编程访问
2. 选项 **2**：编辑 `config/runtime.exs` 并重启应用程序
3. 选项 **3**：使用 Web UI 进行单个路由更改

导出/导入

注意：Web UI 目前不支持导出或导入路由表。路由：

- 存储在 Mnesia 数据库文件中
- 在 `config/runtime.exs` 中配置

要备份路由：

1. **Mnesia**：备份 `Mnesia.{node_name}/` 目录
2. 配置：版本控制 `config/runtime.exs`

自动刷新行为

不同页面具有不同的刷新间隔：

页面	刷新间隔	原因
路由管理	5 秒	路由更改不频繁
活动用户	2 秒	用户状态频繁变化
M3UA 状态	按页面变化	连接状态监控

WebSocket 连接：所有页面使用 Phoenix LiveView WebSocket 连接进行实时更新。

网络中断：如果 WebSocket 连接丢失，页面将尝试自动重新连接。

故障排除

页面未加载

1. 检查 **HTTPS** 证书：确保 `priv/cert/omnitouch.crt` 和 `.pem` 存在
2. 验证端口 **443**：检查防火墙规则允许 HTTPS 流量
3. 应用程序运行：确认应用程序正在运行，使用 `iex -S mix`
4. 浏览器控制台：检查 SSL 证书错误（自签名证书警告）

路由未持久化

1. 检查 **Mnesia** 存储：验证配置中的 `mnesia_storage_type: :disc_copies`
2. **Mnesia** 目录：确保 Mnesia 目录可写
3. 检查日志：查找应用程序日志中的 Mnesia 错误

自动刷新不起作用

1. **WebSocket** 连接：检查浏览器控制台中的 WebSocket 错误
2. 网络：验证网络连接稳定
3. 页面重新加载：尝试刷新页面 (F5)

相关文档

- **STP** □南 - 详细的路由配置
- **HLR** □南 - 用户管理
- **API** □南 - 用于编程访问的 REST API
- 配置参考 - 所有配置参数

总结

OmniSS7 Web UI 提供直观的实时路由表和用户跟踪管理：

□ **□□□□** - 自动刷新保持数据最新

□ **□□□□** - Mnesia 确保路由在重启后仍然存在

□ **□□□□□ UI** - 页面根据操作模式（STP/HLR/SMSc）进行调整

□ **□□□□□** - 添加、编辑、删除路由无需重启

□ **□□□□** - 实时连接和对等状态

有关高级操作或自动化，请参阅 API 指南。